Vol.9 No.6 November 1990

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

Inside the Mandelbrot Set

- THE ARCHIMEDES A540
- WORDPROCESSOR STYLE INPUT
- CURLY EDUCATIONAL GAME
- BETTER PROGRAMMING

BEEBUG

Vol.9 No.6 November 1990

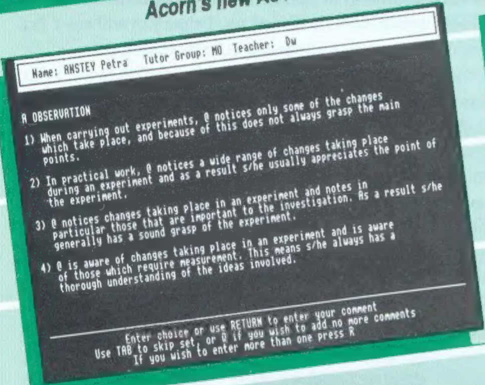
PROGRAM INFORMATION

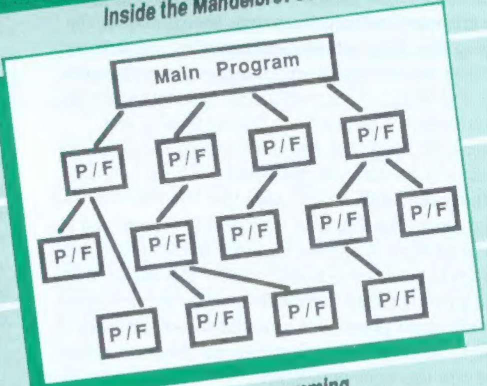
All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly as printed, and checked carefully. When entering a listing, pay special attention to the difference between the digit one and a lower case l (L). Also note that the vertical bar character (Shift \) is reproduced in listings as |.

All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints

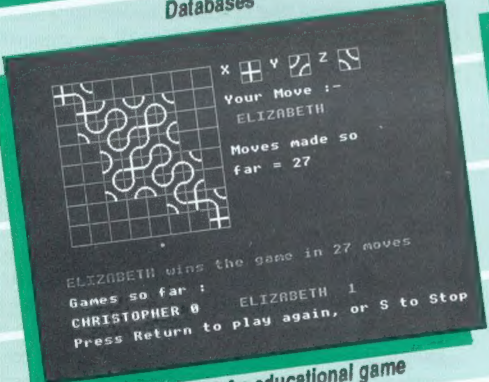
Acorn's new A540

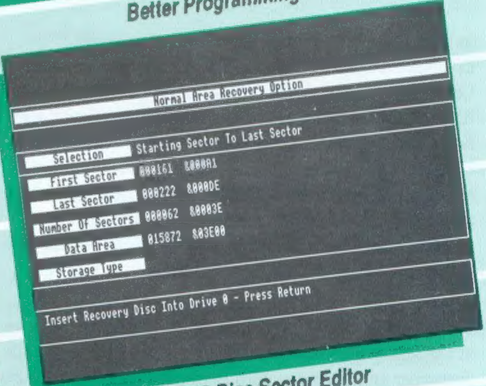
```
   1
-2.00000
2.50E0
-1.25000
2.50E0
      0
      4
      8
     12

>OSCLI"S
AVE FRED
 FFFF300
0 +5000"
```
Inside the Mandelbrot Set


```
Name: ANSTEY Petra   Tutor Group: MO  Teacher:  Dw

A OBSERVATION

1) When carrying out experiments, @ notices only some of the changes
   which take place, and because of this does not always grasp the main
   points.

2) In practical work, @ notices a wide range of changes taking place
   during an experiment and as a result s/he usually appreciates the point of
   the experiment.

3) @ notices changes taking place in an experiment and notes in
   particular those that are important to the investigation. As a result s/he
   generally has a sound grasp of the experiment.

4) @ is aware of changes taking place in an experiment and is aware
   of those which require measurement. This means s/he always has a
   thorough understanding of the ideas involved.

          Enter choice or use RETURN to enter your comment
   Use TAB to skip set, or @ if you wish to add no more comments
          If you wish to enter more than one press R
```
Databases



Better Programming


```
                 X   Y   Z
                 Your Move :-
                  ELIZABETH

                 Moves made so
                 far = 27

ELIZABETH wins the game in 27 moves
Games so far :
CHRISTOPHER 0        ELIZABETH  1
Press Return to play again, or S to Stop
```
Curly! An educational game


```
              Normal Area Recovery Option

Selection          Starting Sector To Last Sector
First Sector       000161  &000A1
Last Sector        000222  &000DE
Number Of Sectors  000062  &0003E
Data Area          015872  &03E00
Storage Type

Insert Recovery Disc Into Drive 0 - Press Return
```
ADFS Disc Sector Editor

available on receipt of an A5 SAE), and are strongly advised to upgrade to Basic II. Any second processor fitted to the computer should be turned off before the programs are run.

Where a program requires a certain configuration, this is indicated by symbols at the beginning of the article (as shown opposite). Any other requirements are referred to explicitly in the text of the article.

	Program will not function on a cassette-based system.
	Program needs at least one bank of sideways RAM.
	Program is for Master 128 and Compact only.

Editor's Jottings

EVENTS FOR ENTHUSIASTS

The calendar seems packed this autumn with events for all those interested in Acorn computers, though it has to be admitted that many software and hardware suppliers are these days increasingly concentrating on the Archimedes range. Following our successful Open Day in October, we shall be holding a further Open Day on Sunday 25th November. Although this will provided a good opportunity to make all your computer purchases, with the help and guidance of our staff, we also hope that members will treat the day as an opportunity to meet BEEBUG's staff and to see how we work. We on the magazine would certainly be delighted to meet and talk with you. We also expect to be running a competition in conjunction with our next Open Day, with major prizes such as a Star colour printer for the lucky winners. More information on this event will be found with this issue of BEEBUG.

In December, 6th to 9th to be precise, the second *Computer Shopper Show* will take place at the Wembley Conference Centre. A self-contained area of this exhibition has been set aside for Acorn and supporting stands, and Acorn is organising a series of seminars for each day which sound most inviting. BEEBUG will have its own stand in this area of the show, with a full range of products and staff to help. We have also arranged to include in the December issue of BEEBUG a voucher giving you a discount off the standard entrance price.

The All Formats Computer Fairs also continue, with shows at the New Horticultural Hall Westminster on Sunday 4th November and Saturday 15th December. Readers will also see from this month's News page details of other regional or specialised shows and other events which are taking place. It certainly looks like being a busy season in the Acorn World.

BEEBUG SURVEYS

We have been very pleased with the response to BEEBUG Surveys, which have so far covered word processors and databases. What we have set out to do in each case is to find experienced and enthusiastic users of individual packages who are able to write an account of the software, and their experiences in using it.

We need your help to pursue the programme of future surveys which we have drawn up. Below you will see a list of some of the subjects we expect to be covering in forthcoming issues. If you consider yourself to be reasonably experienced and competent with any particular product which fits into this list then we would like to hear from you. Please write to us saying which products you would be prepared to write about, what experience you have in using them, and convince us that you could do a good job of writing about this in an informative and interesting style. Please make sure that you include a daytime phone number where possible.

Possible future topics are:

- Spreadsheets
- Creative Graphics (Art)
- Technical Graphics (CAD)
- Printer Dump Utility ROMs
- General Utility ROMs

We are also quite prepared to consider suggestions for surveys in other areas, including hardware.

ACORN'S FLAGSHIP

We recently had the opportunity to visit Acorn in Cambridge and spend a day trying out the new Archimedes A540, which we mentioned in last month News page. We believe that BEEBUG readers will want to keep up to date with major events in the Acorn world so we make no apologies at all for devoting two pages in this issue to this super fast machine. The quality of screen image which can be generated by this system is superb, and coupled with almost unbelievable processing power this is certainly a machine to covet. Unfortunately, at a price of nearly £3000 (and that's before VAT) dreaming about this system is as close to owning such a machine that most of us will ever manage. Nevertheless, we should all applaud Acorn for the technical innovation of this flagship in the Archimedes range.

News News News News News News

COMPUTER SHOW FOR SCOTLAND

The SATRO (Science And Technology Regional Organisation) 4th annual show will take place at Aberdeen Music Hall on 9th December 1990 from 11am to 5pm. This is the North of Scotland's largest gathering of computer and science enthusiasts, bringing together schools, colleges, universities and other related groups to present the fruits of their endeavours and to help and encourage others. Trade stands will also be present for the sale of computers and related items. Entrance costs £1.00 (50p concessions). For further information contact SATRO North Scotland, Marischal College, University of Aberdeen, Broad Street, Aberdeen AB9 1AS, tel. (0224) 23161.

COMPUTERS FOR THOSE WITH SPECIAL NEEDS

SNUG (Special Needs User Group) exists to help and support all those interested in using computers to aid those with handicaps or other special needs. SNUG provides a telephone help line for those who need advice and information, but would also welcome support and interest from those who would like to contribute in some way to the aims of SNUG. Membership currently costs £4 per annum, and more information on all of SNUG's activities can be obtained from the secretary, Jeff Hughes, 39 Eccleston Gardens, St Helens WA10 3BJ, tel. (0744) 24608.

STUDENT LOAN SCHEME

Not the government's controversial student loan scheme, but a new opportunity for students at college and university to obtain an Archimedes computer system on the cheap. Acorn has joined forces with NUS Services Ltd and Barclays Bank to provide a unique 'futures' computer purchase scheme. Students taking advantage of the scheme get free use of a computer while they study, and only commence repayments once they start earning, and at preferential rates.

By this means, Initial Teacher Training students can obtain an Acorn *Learning Curve* (A3000-based) system for just £977.85 inc. VAT. An alternative system is the A420/1 (with 2MBytes of memory and 20MByte hard disc) which will have Acorn DTP software bundled in at a special price of £1683.00 inc. VAT (price under review).

Full details of the new scheme are included in the NUSSL catalogue *nsm Computing Extra* distributed through universities, polytechnics and colleges, or available direct from NUS Services Ltd on (0457) 868003.

MUSIC 5000 LEADS THE WAY

Hybrid Technology's Music 5000 continues to lead the music market for the BBC micro and Master series. Latest developments include new Music 5000 Synthesiser Universal versions of existing music software for the Master 128. These are *Rhythm Maker* and *Compose Play* from ESP, *Basic Music Composition* from Tobin Music Systems (see review in BEEBUG Vol.9 No.5), and *Theory of Music Tutorial* and *Theory of Music Questions* from Ted Kirk.

New music compositions available for the Music 5000 are *Ashes* by Michael Harbour, *AMPLE DCT* by Alistair Johnston and *Impressions* by John Bartlett, and all available from Hybrid.

Addresses for more information:

Hybrid Technology Ltd., 273 The Science Park, Cambridge CB4 4WE, tel. (0223) 420360.
ESP, Holly Tree Cottage, Main Street, Strelley Village, Nottingham NG8 6PD, tel. (0602) 295019.
Tobin Music Systems, The Malthouse, Knight Street, Sawbridgeworth, Herts CM21 9AX, tel. (0279) 726625.
Ted Kirk, 33 Humber Crescent, Sutton Leach, St Helens, Merseyside WA9 4HD, tel. (0744) 818761.

NEW COMPUTER GROUP FOR DUBLIN

A new special Interest Group for IBM and Acorn computers has been formed in Dublin, in the Republic of Ireland. For more information write to The Special Interests Group, P.O. Box 2934, Dublin 22, Ireland.

EDUCATIONAL CONFERENCE AND EXHIBITION

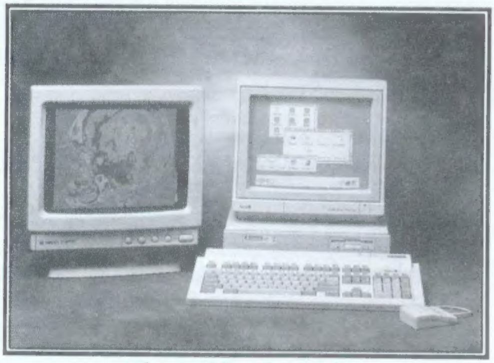
The Resource Conference, *Information Technology Across the National Curriculum* takes place on Thursday 15th November at the Doncaster Exhibition Centre. An exhibition with over 70 participating companies and organisations will accompany the day's programme of events. For more details contact Resource, Exeter Road, Doncaster, South Yorkshire DN2 4PY, tel. (0302)

Acorn's New Archimedes Flagship: the A540

Reviewed by Mike Williams and Alan Wrigley

As reported in last month's BEEBUG, Acorn has released a new top-of-the range Archimedes known as the A540 model. Here we present an assessment of this very fast machine.

Externally there is little to distinguish the A540 from the existing A400/1 range, apart from the model number on the front. However, a second glance at the keyboard shows that the folding transparent keystrip holder has disappeared, leaving a clean raised surface running the whole width. Otherwise the keyboard appears identical to existing models, but Acorn says that production versions will use the same mechanism as on the A3000.



The Archimedes A540

At the rear there is a SCSI connector (on the rear of the SCSI interface card occupying one of the expansion card slots on the A540). The SCSI interface supports an internally mounted SCSI hard disc drive (an integral part of the A540) of nominal 100MByte capacity. The specification for this drive indicates an average seek time of 25ms. This gives the A540 high speed, large capacity storage. Internally, provision has also been made for Genlock and similar video digitiser boards to be connected directly into the Archimedes.

Removing the lid reveals the floppy and SCSI disc drives, the power supply, and the SCSI interface, (which leaves room for just three further expansion cards). Underneath all this is the newly designed A540 main board, which now provides video support for high resolution VGA and super VGA (SVGA), of which more later. Four sockets to the front of the backplane position will accommodate up to 16MBytes of memory, though just 4MBytes is supplied as standard. Each memory board will contain its own MEMC (memory controller) chip.

The ARM3 processor is now mounted on a separate plug-in board which fits next to the memory board, and so close to the backplane (which protects it) that it might be overlooked. The ARM3 processor is designed to run at 30MHz, and one of the major factors in the boost to performance. The use of a separate plug-in processor board also indicates Acorn's desire to provide better opportunities for upgrading the processor in the future.

In addition, the processor board comes with a socket ready for the insertion of a new Floating Point Accelerator chip (FPA) as a further boost to performance (Acorn suggested by a factor of 10 in appropriate circumstances for FPA alone). However, the FPA is not scheduled for release until the summer of 1991, and is intended to be a cost option for the A540.

Not only is the ARM3 processor significantly faster than the ARM2 currently used, but also sports an internal 4KByte cache memory. Fast cache memory holds previously used data and instructions (which might be used again), and significantly improves throughput. The difference is quite noticeable when refreshing the Desktop display, for example. Indeed, some games might become unplayable with the cache switched on.

The A540 is supplied with the same software as the current A400 series with the addition of some new modules to support the ARM3 including the 4K cache, the VGA and SVGA standards referred to before, and the SCSI interface. All other software, including RISC OS will be as supplied with other Archimedes machines.

To support the VGA and SVGA standards (used by PCs) there are five new modes as shown in table 1. These modes are also supported by multi-sync monitors, such as the Eizo or Taxan 770. However, comparatively cheap VGA and SVGA monitors designed for use with PC compatibles can be used as alternatives. If required, the A540 will still support a standard Archimedes monitor.

In use the picture quality achievable with VGA and SVGA is quite staggering. Full colour, photographic quality pictures are now possible, and are outstanding in terms of resolution and colour realism. Indeed, looking at some of the full screen sample pictures it is hard to believe that you are not looking at a photograph, or original colour print.

Mode	Stnd	Colours	Memory	Pixels
25	VGA	2 colour	64K	640x480
26	VGA	4 colour	96K	640x480
27	VGA	16 colour	160K	640x480
28	VGA	256 colour	320K	640x480
31	SVGA	16 colour	256K	800x600

Table 1. New VGA and SVGA modes

PERFORMANCE

How fast is the new A540? There are a number of ways to answer this question. First of all, with the 30MHz ARM3, 12MHz memory sub-system and 8MHz I/O the A540 offers a throughput of 13.5 MIPs (Millions of Instructions Per second). That compares with 4.55 MIPs for a PC AT (286 CPU at 10MHz) and 9.59 MIPs for a PS/2 70 (386 CPU at 25MHz).

To try and provide a quantitative comparison between the A540 and other machines in the Archimedes range we ran a number of simple benchmarks, the results of which are shown in table 2. These figures also show the effect of using cache memory on the A540.

For reference, RealTest involved multiplying two reals, StringTest writing a string to memory, and ScreenWrite writing a string to screen. The A540 was tested both with cache on and with cache off. Notice the often significant difference in timings here. Modes 12 and 15 in all cases were with a standard monitor, new modes 28 and 31 with a multi-sync. It may be worth noting that a 400 series machine fitted with an ARM3 upgrade (which also includes cache memory) also runs very fast, but cannot quite match the A540. The NewModes module will also be available separately from Acorn for those who wish to use the new modes on existing machines.

Mode	12	15	28	31
RealTest (10,000)				
A310	178	221	—	—
A540 (off)	97	114	199	145
A540 (on)	27	28	32	29
StringTest (10,000)				
A310	974	1232	—	—
A540 (off)	555	653	1150	830
A540 (on)	187	194	230	193
ScreenWrite (1,000)				
A310	183	243	—	—
A540 (off)	161	201	351	245
A540 (on)	61	72	98	72

Table 2. Times for benchmarks (in centi-secs)

SUBJECTIVE PERFORMANCE

During extensive use of the A540, normally with cache on, we were continually impressed by the speed of the system. For DTP work, scrolling a 20 page DTP document caused no delays at all, and sprites imported into picture frames could be moved around on the page, with reflow of text, very quickly indeed. The A540 is clearly a superb machine for DTP work, and with the choice of software now available compares very favourably with PCs and the Apple Macintosh in this field.

The A540 is equally impressive in graphics work and animation. With the SCSI disc, captured video sequences can be shown in real time, and images processed and manipulated with commendable speed. General Desktop screen work happens so fast that it is a real pleasure to use the system.

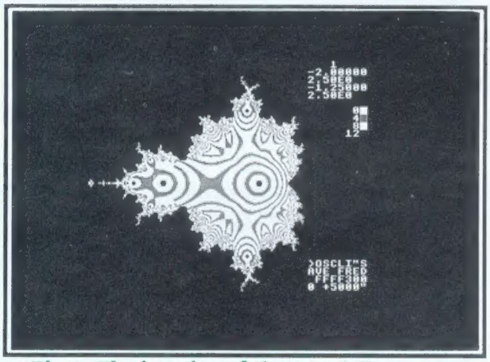
CONCLUDING COMMENTS

The A540 is an outstanding machine, but at a price of £2995 plus VAT is for the moment probably beyond the pockets of most home users, more so when the cost of a suitable multi-sync or VGA monitor is included. It is also worth noting that the A540 can be turned into an Acorn Unix machine merely by installing the relevant software (though Acorn has no plans at present to make this available separately). The A540 certainly offers outstanding performance for the price, and one can only hope that as with DTP, and increasingly for image processing, video digitising and the like, the software and hardware will appear which will truly exploit this machine to the full.

Inside the Mandelbrot Set

We make no apology for returning to the subject of the Mandelbrot Set in this splendidly updated program by Kate Crennell.

Many programs have been published which plot views of the exterior of the Mandelbrot Set, including *Mandel* in BEEBUG Vol.5 No.1 (May 1986). The program listed here, *Mandel2*, plots views of the *interior* of the Set, as well as allowing any power law between 2 and 99, and having a simple menu interface.



```
    1
-2.00000
2.50E0
-1.25000
2.50E0

      0
      4
      8
     12

>OSCLI"S
AVE FRED
 FFFF300
0 +5000"
```

Fig.1 The interior of the Mandelbrot set

RUNNING INSTRUCTIONS

Type in the program and save it as *Mandel2*. Take care not to add any extra spaces, and omit the space between the line number and the following instruction, which has been included for readability. When you run it, you see a title page, followed by a menu screen showing the names and values of all the parameters, with a flashing cursor adjacent to the word 'plot'. The default plot is the familiar shape of the Mandelbrot Set using a power law of 2. Either press the Return key to start plotting using these values, or use the cursor keys to move the flashing cursor adjacent to the name of the parameter you want to change. Then when you press the Return key the name of the parameter you are changing is displayed in the screen window on the right, and you are asked to enter a new value. Use a small number of rows and columns to see a low resolution plot quickly. Plots always use square pixels, but their size depends on how many rows and columns you asked for.

Try using 32 rows and 32 columns, and plot this default to check that it looks familiar. The plot is square, with the plot area listed upper right, the bounds and colours below, and the number of the row being plotted at the bottom.

Then try setting the power law to -2, for a plot of the interior of the Set. This should resemble Figure 1 with striped bands inside the Set and the outside left blank. Interior plots can be made for laws -2, -3, and -4, exterior plots for any value of law between 2 and 99. Values above 4 all plot at the same speed. Figure 2 shows a mode 1 plot for power law 7.

You change the number of bounds by moving the cursor adjacent to a colour, pressing Return, and then moving the cursor over the menu in the screen window to select 'Insert', 'Delete' or 'Change the Colour'. To change the number associated with a bound, put the cursor adjacent to the number instead of the colour, before pressing Return.

When the plot is finished, a beep sounds, and another small menu appears at the bottom right of the screen. Pressing the Escape key during plotting also sends the program to this point. Use the cursor keys to position the line shown in inverse video, and then press the Return key for action. Possible choices are to save the screen, return to the main menu, or ask for a 'box' to define an area of the plot for replotting at a higher magnification. The box is drawn in the centre of the plot initially, and this can be moved using the cursor keys. The + key makes a bigger box, - a smaller one. Press M for the main menu.

HOW TO SEE THE INSIDE OF THE SET

I got this idea from a new journal, *Algorithm*, the personal programming newsletter, editor

A.K.Dewdney, who also writes the *Computer Recreations* section of *Scientific American*. The journal describes algorithms in simple terms, so that owners of home computers can understand how to program the algorithm for themselves. It does not give complete programs for particular computers. Subscription details are given at the end of this article. In the first issue Clifford Pickover explains how to plot variations on the usual Mandelbrot Set, with an illustration similar to Figure 1.

The interesting regions of concentric stripes surround each 'centre of attraction' of the set. These are mathematical features, sometimes called the 'zeros' of the set. Near the edge of the set these get closer together, and surround the 'buds' which grow out on the sides. There are many interesting plots of internal regions waiting to be found. Start with the default values for the three power laws, and use the box to define new places, or type in co-ordinates.

In the usual plots the iteration formula is:

```
Z ← Z^2 + c
```

which is calculated until Z exceeds a given maximum, when the number of iterations reached is used to decide what colour to plot for that point. Inside the set, the maximum can never be reached so this area is usually shown black.

If instead of keeping the highest iteration count reached, we keep the smallest value Z reaches, another plot can be drawn, with the colour of each point showing the smallest Z at that point. The only problem is how to show small continuous values as a discrete small set of colours. I have chosen an arbitrary value of 100 for the starting value of the variable *Small* used to store the smallest value in procedure PROCi, which calculates the formula for the interior of the set for the 2nd, 3rd and 4th power laws. This value needs to be converted into a number in the range 0 to 7 for plotting on the BBC micro. This is done at line 1270, adjusted for power laws 2, 3, and 4 by the value of T% (which holds this value).

Here are a few interesting data sets:

Law	Xcentre	width	Ycentre	height
-2	0.347	0.2	0.097	0.2
-2	-0.5046	0.0111	-0.6117	0.0111
-3	-0.024	0.69	1.016	0.69
-4	-0.21322	0.00184	0.83886	0.00184

You might like to try other algorithms for displaying the colours, or change procedure PROCi to save other things such as the largest value reached instead of the smallest.

HIGHER POWER LAWS

The idea for the higher power laws came from another newsletter, *Fractal Report* where Andy Luness in issue 8 explained how to plot any power law. I have coded the laws 2, 3 and 4 individually, and placed the procedure Mandel near to the beginning of the program for increased speed. The higher power laws (>5) are all plotted by the single procedure, PROCAny. All higher power law plots take about the same time. Notice that in Figure 2, a plot of the 7th power law has 6 axes of symmetry. It is a general rule that the plot of the complete set for the nth power law has (n-1) axes of symmetry. You might try a few other laws to check this.

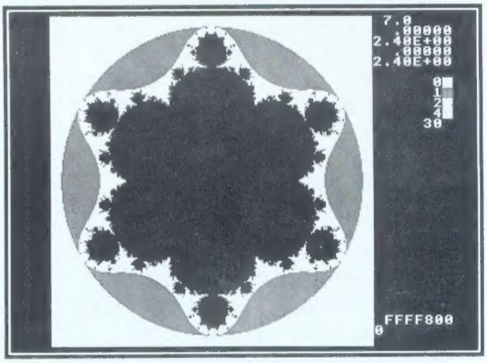


Fig.2 Mandelbrot set using a power level of 7

Fractional power laws are also possible, but this needs more changes because the law is stored in the integer variable, T%, which would need to be changed to a real variable throughout the program.

NEWSLETTER SUBSCRIPTION DETAILS

A year's subscription to *Algorithm* costs $30 + $4 (US$) for overseas shipping from Algorithm, PO Box 29237, Westmount Postal Outlet, 785 Wonderland Road, South London, Ontario, Canada N6K 1M6.

A subscription to *Fractal Report* costs £10.00 (UK), £12.00 (Europe) from Reeves Telecommunications Laboratories Ltd, West Towan House, Porthtowan, Truro, Cornwall TR4 8AX.

Fractal Report is a newsletter for all those interested in fractals; it is not specific to a particular computer. The editor prefers short programs in Basic which demonstrate the ideas. He offers free subscriptions to a future volume, (six issues), to any contributor of an article he publishes.

```
   10 REM Program Mandel2
   20 REM Version B1.2
   30 REM Author  Kate Crennell
   40 REM BEEBUG  November 1990
   50 REM Program subject to copyright
   60 :
  100 DIMc%(7),OP$(3),d%(15),k%(15)
  110 ON ERROR GOTO250
  120 IF PAGE<>&1100 THEN PAGE=&1100:CHA
IN"MANDEL2"
  130 MODE7:PROCIntro
  140 MODE7:L%=2:PROCAll
  150 E%=1:PROCMenu:MODEM%:IFM%=1FORI%=0
TO3:VDU19,I%,c%(I%)MOD16;0;:NEXT
  160 VDU28,G%,31,H%,0:@%=4:PRINTT%:@%=&
20508:PRINTX;:@%=&10308:PRINTW;:@%=&2050
8:PRINTY;:@%=&10308:PRINTH:@%=11-M%*4:FO
RI%=0TOC%:PRINTd%(I%);:COLOURk%(I%):VDU2
55:COLOUR7:NEXT:@%=8/M%
  170 E%=2:VDU23,1,0;0;0;0;28,G%,31,H%,2
9
  180 R=X+(DX-W)/2:S=Y+(DY-H)/2
  190 PROCMandel(R,S,DX,DY)
  200 VDU7:PROCPick:IFP%>0GOTO140
  210 CLS:INPUT"Filename";,F$
  220 IFM%=1 FORI%=0TO3:I%?&7FFC=c%(I%)M
OD16:NEXT
  230 OSCLI"SAVE "+F$+" 3000 8000"
  240 GOTO200
  250 IFE%<>2 OR ERR<190 GOTO270
  260 CLS:REPORT:A$=GET$:GOTO200
  270 IFERR<>17 MODE7:REPORT:PRINT;" AT
LINE ";ERL:OSCLI"FX4":END
  280 ONE%+1GOTO290,150,200
  290 PROCq
  300 :
 1000 DEFPROCMandel(Xmin,Ymin,Dx,Dy):O%=
d%(C%)
 1010 u%=4*M%*U%:v%=4*V%:u1%=u%-4:v1%=v%
-4
 1020 IFT%>4 Limit=(2^(1/(T%-1)))^2:F=T%
/2
 1030 y=Ymin-Dy:FORy%=0TOY%:PRINTy%:y=y+
Dy:x=Xmin-Dx:FORx%=0TOX%:x=x+Dx:U=x:V=y:
N%=0
 1040 IFT%>0 AND T%<5 exit=FALSE:REPEAT:
A=U*U:B=V*V:ONT%-1GOSUB1090,1110,1130:N%
=N%+1:UNTILN%>O%ORexit ELSE PROCMani
 1050 J%=k%(C%):FORI%=C%TO1STEP-1:IFN%<d
%(I%) J%=k%(I%-1)
 1060 NEXT:IFJ%=0 GOTO1080
 1070 GCOL0,J%:IF U%>1 MOVEx%*u%,y%*v%:M
OVEx%*u%+u1%,y%*v%:PLOT85,x%*u%,y%*v%+v1
%:MOVEx%*u%+u1%,y%*v%:PLOT85,x%*u%+u1%,y
%*v%+v1% ELSE PLOT69,x%*u%,y%*v%:IFM%=2P
LOT69,x%*u%,y%*v%+4
 1080 NEXT,:ENDPROC
 1090 IFA+B>4 exit=TRUE:RETURN
 1100 V=y+2*U*V:U=x+A-B:RETURN
 1110 IFA+B>2 exit=TRUE:RETURN
 1120 V=y+V*(3*A-B):U=x+U*(A-3*B):RETURN
 1130 IFA+B>2 exit=TRUE:RETURN
 1140 V=y+4*U*V*(A-B):U=x+A*A+B*B-6*A*B:
RETURN
 1150 :
 1160 DEFPROCMani:Small=100.
 1170 IFT%>4 PROCAny ELSE IF T%>-5 AND T
%<-1 PROCi
 1180 ENDPROC
 1190 :
 1200 DEFPROCi:exit=FALSE
 1210 REPEAT:A=U*U:B=V*V:C=A+B
 1220 IFC<Small Small=C
 1230 C1=2-2*(T%=-2):IF C>C1 N%=O%:exit=
TRUE:GOTO 1260
 1240 ON T%+5GOSUB 1330,1320,1310
 1250 N%=N%+1
 1260 UNTIL N%=O%ORexit:IF exit ENDPROC
 1270 s=Small*100*(2+(T%=-4))
 1280 IF s<0.5 N%=O% ELSE N%=s MODO%
 1290 ENDPROC
 1300 :
 1310 V=y+2*U*V:U=x+A-B:RETURN
 1320 V=y+V*(3*A-B):U=x+U*(A-3*B):RETURN
 1330 V=y+4*U*V*(A-B):U=x+A*A+B*B-6*A*B:
RETURN
```

```
 1340 :
 1350 DEFPROCAny:exit=FALSE
 1360 REPEATA=U*U:B=V*V:C=A+B:IF C>Limit
 exit=TRUE:GOTO1400
 1370 IF U>0 T=ATN(V/U) ELSE IFU<0 THEN
T=PI+ATN(V/U) ELSE T=PI*.5*SGN(V)
 1380 T=T*T%:C=C^F:U=x+C*COST:V=y+C*SINT
 1390 N%=N%+1
 1400 UNTIL N%>O%ORexit
 1410 ENDPROC
 1420 :
 1430 DEFPROCIntro:REM Introduction
 1440 OP$(0)="Save":OP$(1)="Menu":OP$(2)
="Stop":OP$(3)="Box"
 1450 E%=0:L%=1:M%=0:C$=CHR$131:w$=CHR$1
35:*FX4,1
 1460 VDU23,255,255,255,255,255,255,255,
255,255
 1470 FORI%=4TO5:PRINTTAB(10,I%)CHR$141+
C$+CHR$157+CHR$132+"MANDELZOOM  "+CHR$15
6+w$:NEXT
 1480 PRINTTAB(10,12)"Version 2 Aug 1990
 "TAB(5,17)"Any power law outside the se
t"''" square, cubic and quartic laws ins
ide":PROCAll
 1490 PRINTTAB(5,23)"Press"+C$+CHR$157+C
HR$132+"Spacebar  "+CHR$156+w$+"to start
":A$=GET$
 1500 ENDPROC
 1510 :
 1520 DEFPROCAll:REM makes screen menu
 1530 IFL%AND2FORI%=1TO2:PRINTTAB(5,I%)C
HR$141+C$+CHR$157+CHR$132+"PLOT PARAMETE
RS "+CHR$156:NEXT:PRINTTAB(0,8)C$+"Plot
Bound Colour"TAB(21)C$+"Stop"+w$:PRINTTA
B(16,23)"Select with cursor keys"'TAB(16
)"Return for action"
 1540 IFL%AND2FORI%=9TO20:PRINTTAB(15,I%
)CHR$132+CHR$157+w$:NEXT
 1550 PROCMode:PROCLaw:PROCArea:PROCDim:
PROCKols
 1560 ENDPROC
 1570 :
 1580 DEFPROCArea:REM area to be plotted
 1590 IFNOTL%AND1GOTO1640
 1600 IFABS(T%)=2 X=-.75:W=2.5
 1610 IFABS(T%)=3 OR T%>4 X=0:W=3
 1620 IFABS(T%)=4 X=-.25:W=2.5
 1630 H=W:Y=0
 1640 IFNOTL%AND4GOTO1690
 1650 IFA%=2X=FNr("Xcen",X,-2,2)
 1660 IFA%=3Y=FNr("Ycen",Y,-2,2)
 1670 IFA%=4W=FNr("Width",W,0,6):IFW=0GO
TO1670
 1680 IFA%=5H=FNr("Height",H,0,6):IFH=0G
OTO1680
 1690 IFL%AND2@%=&2070A:PRINTTAB(0,4)C$+
"Xcen ="+w$,X,TAB(21)C$"Ycen  ="+w$,Y:@%
=&1040A:PRINTTAB(0,5)C$+"Width="+w$,W,TA
B(21)C$"Height="+w$,H
 1700 IFL%<3ENDPROC
 1710 K%=256/M%:R=H/W:IFR>1Y%=K%:X%=Y%/R
+0.5ELSEX%=K%:Y%=X%*R+0.5
 1720 IFX%>0X%=X%-1
 1730 IFY%>0Y%=Y%-1
 1740 L%=2:PROCDim
 1750 ENDPROC
 1760 :
 1770 DEFPROCBox:CLS:PRINT"M"'"for"'"Men
u"
 1780 S%=4*M%*U%:A%=S%*(X%DIV2):B%=S%*(Y
%DIV2):L%=S%*4-1:GCOL3,M%*4-1:PROCB
 1790 REPEAT
 1800 REPEAT:I%=INSTR(CHR$136+CHR$137+CH
R$139+CHR$138+"+-Mm",GET$):UNTILI%:PROCB
 1810 IFI%=1ANDA%>0A%=A%-S%
 1820 IFI%=2ANDA%<1023-L%A%=A%+S%
 1830 IFI%=3ANDB%<1023-L%B%=B%+S%
 1840 IFI%=4ANDB%>0B%=B%-S%
 1850 IFI%=5ANDL%<1023-S%L%=L%+S%
 1860 IFI%=6ANDL%>S%L%=L%-S%
 1870 PROCB:UNTILI%>6
 1880 I%=1+L%DIV2
 1890 X=X-W/2+DX*(A%+I%)/S%:Y=Y-H/2+DY*(
B%+I%)/S%:W=L%*DX/S%:H=L%*DY/S%:ENDPROC
 1900 :
 1910 DEFPROCB:MOVEA%,B%:MOVEA%+L%,B%:PL
OT85,A%,B%+L%:DRAWA%+L%,B%:PLOT85,A%+L%,
B%+L%:ENDPROC
 1920 :
 1930 DEFPROCDim:REM rows & columns
 1940 IFNOTL%AND1GOTO1980
 1950 IFM%=1X%=255
 1960 IFM%=2X%=127
 1970 Y%=X%
 1980 IFNOTL%AND4GOTO2020
 1990 K%=256/M%:PRINT"Maximum rows or"'"
columns: ";K%
 2000 IFA%=6Y%=FNr("rows",Y%+1,1,K%)-1
 2010 IFA%=7X%=FNr("columns",X%+1,1,K%)-
1
 2020 J%=X%+1:K%=Y%+1
 2030 IFL%AND2@%=3:PRINTTAB(0,6)C$+"No.
of rows"+w$,K%,TAB(21)C$+"No. of cols"+w
$,J%
 2040 I%=256/M%:DX=W/J%:DY=H/K%:IFJ%>K%U
```

```
%=I%DIVJ%ELSEU%=I%DIVK%
 2050 V%=U%*M%
 2060 ENDPROC
 2070 :
 2080 DEFPROCKols:REM bounds & colours
 2090 IFL%AND8GOTO2120
 2100 IFNOTL%AND1GOTO2140
 2110 C%=7:c%(0)=0:c%(1)=7:FOR I%=0TO15:
d%(I%)=2*I%:NEXT
 2120 IFM%=2k%(0)=4:k%(1)=1:k%(2)=5:k%(3
)=2:k%(4)=6:k%(5)=3:k%(6)=7:k%(7)=0:GOTO
2140
 2130 IFM%=1 FOR I%=0TO15:k%(I%)=(I%+1)M
OD4:NEXT:c%(1)=1:c%(2)=2:c%(3)=7:FORJ%=4
TO7:c%(J%)=J%-1:NEXT
 2140 IFNOTL%AND4GOTO2350
 2150 B%=A%DIV2-5:IFB%=0 Q%=0 ELSEQ%=d%(
B%-1)+1
 2160 IFB%=C% R%=32767 ELSE R%=d%(B%+1)-
1
 2170 IFA%AND1GOTO2200
 2180 IFB%=0VDU26:ENDPROC
 2190 PRINT"allowed values"'"from ";Q%"
to ";R%:d%(B%)=FNr("boundary",d%(B%),Q%,
R%):GOTO2350
 2200 N%=9:K%=d%(B%):PRINT'''"At bound "
;K%'''''"select colour,"'"'Delete' or"'"
'Insert' band.":IFR%=K%ORC%=15N%=8
 2210 FORI%=0TON%:K%=I%:IFM%=1ANDI%<8K%=
c%(K%)
 2220 PRINTTAB(16,I%+1)CHR$156;:IFI%=8PR
INT"Del"ELSEIFI%=9PRINT"Ins"ELSEIFK%>0PR
INTCHR$(128+K%)+CHR$255
 2230 NEXT:I%=0:REPEAT:VDU31,16,I%+1:J%=
GET:IFJ%=139I%=I%-1:IFI%<0I%=N%
 2240 IFJ%=138I%=I%+1:IFI%>N%I%=0
 2250 UNTILJ%=13:IFI%<8GOTO2320
 2260 IFI%=9C%=C%+1:FORI%=C%TOB%+1STEP-1
:d%(I%)=d%(I%-1):k%(I%)=k%(I%-1):NEXT:d%
(B%+1)=R%:GOTO2340
 2270 IFC%<2GOTO2330
 2280 IFC%<2GOTO2340
 2290 CLS:VDU26:C%=C%-1:PRINTTAB(1,C%+9)
SPC14:IFB%<C%+1:FORI%=B%TOC%:d%(I%)=d%(I
%+1):k%(I%)=k%(I%+1):NEXTELSEA%=A%-2
 2300 IFB%=0 d%(0)=0
 2310 GOTO2350
 2320 IFM%=2ORI%<4 k%(B%)=I%:GOTO2340
 2330 J%=k%(B%):K%=c%(J%):c%(J%)=c%(I%):
c%(I%)=K%
 2340 CLS:VDU26
 2350 IFNOTL%AND2GOTO2390
 2360 @%=6:FORI%=0TOC%:K%=k%(I%):IFM%=1K
%=c%(K%)
 2370 PRINTTAB(4,I%+8)d%(I%);:IFK%>0PRIN
T" "CHR$(128+K%)+CHR$255+w$ELSEPRINTSPC3
CHR$132
 2380 NEXT
 2390 ENDPROC
 2400 :
 2410 DEFPROCLaw:REM law 2 to 99
 2420 IFL%AND1T%=2:REM default law 2
 2430 IFL%AND4REPEAT:VDU28,18,20,39,9,30
:PRINT"Possible power laws"'"are 2 to 99
"'" or for internal"'"plots -2,-3,-4":T%
=FNr("power law",T%,-4,99):UNTIL ABS(T%)
>1 :L%=3:PROCArea
 2440 IFL%AND2@%=3:PRINTTAB(21,3)C$+"Law
:"+w$+"w^";T%;"+Z-> w'"
 2450 ENDPROC
 2460 :
 2470 DEFPROCMenu:VDU23;10,98;0;0;0:REM
Controls Menu
 2480 A%=8
 2490 Z%=C%*2+11:REPEAT:IFA%<10VDU31,21*
(A%MOD2),3+A%DIV2ELSEVDU31,10+(A%MOD2),3
+A%DIV2
 2500 I%=GET
 2510 IFI%=136A%=A%-1
 2520 IFI%=137A%=A%+1
 2530 IFI%=138A%=A%+2
 2540 IFI%=139A%=A%-2
 2550 IFA%<0A%=A%+Z%+1
 2560 IFA%>Z%A%=A%-Z%-1
 2570 UNTILI%=13
 2580 L%=6:VDU28,18,20,39,9,30
 2590 IFA%>9PROCKols
 2600 IFA%=0PROCMode
 2610 IFA%=1PROCLaw
 2620 IFA%>1ANDA%<6PROCArea
 2630 IFA%=6ORA%=7PROCDim
 2640 IFA%=8ENDPROC
 2650 IFA%=9PROCq
 2660 GOTO2490
 2670 :
 2680 DEFPROCMode:REM mode changes
 2690 IFL%AND1M%=2
 2700 IFL%AND4PRINT"Resolution of"'"Mode
 1 is 256 lines"'"      2 is 128 lines":M
%=FNr("plot mode",M%,1,2):L%=3:PROCDim:L
%=10:PROCKols
 2710 IFL%AND2@%=3:PRINTTAB(0,3)C$+"Scre
en mode"+w$;M%
 2720 G%=32/M%:H%=59-20*M%:REM default w
indow sizes
 2730 ENDPROC
```

continued on page 55

BEEBUG Survey: Databases

This month we complete our survey of databases for the BBC micro with a look at one of the more recent packages to appear, *InterBase*, and one of the oldest, *Beta-Base*.

InterBase

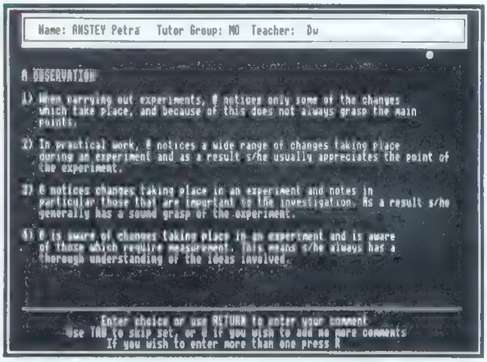
John Mawer explains why he believes Computer Concepts' database is so powerful.

InterBase (Computer Concepts) £56.35
(BEEBUG retail price inc. VAT)

InterBase was the last of the Inter series to be produced by Computer Concepts. Those of us who grew up on InterWord, InterSheet and InterChart anticipated yet another ingenious piece of software to take the BBC to its limits. And so it was to be, but not without the now familiar Computer Concepts delays and a false start with an exasperating first manual. Yet anyone who weathered these initial frustrations now has access to a programmable database with facilities way beyond anything originally dreamt of on the BBC.

The software is ROM based, using the now familiar 'piggy back' 64K ROM. The package comes with a much improved manual and a disc of example programs. For those who expect only to use the card index example program included in the ROM, InterBase will seem no different from many other packages. It is after all, only an example database, and many of the features of InterBase are left unused.

InterBase contains an extremely powerful language. It is similar to Basic, but its control structures such as WHILE .... ENDWHILE loops, and CASE .... ENDCASE, make it closer to Basic V as used on the Archimedes than the earlier versions. The standard arithmetic functions .turn operations such as standardising test marks and sorting into rank order into trivial tasks. String manipulation facilities are considerably more extensive than in Basic, and with the inclusion of key words like WORD$ and LINE$, operations such as specifying words within a string can be tackled with ease. It is in the area of string handling that it is at its most powerful, so I will return to this later.



```
Name: ANSTEY Petra   Tutor Group: MO  Teacher:  Dw

A OBSERVATION

1) When carrying out experiments, @ notices only some of the changes
   which take place, and because of this does not always grasp the main
   points.

2) In practical work, @ notices a wide range of changes taking place
   during an experiment and as a result s/he usually appreciates the point of
   the experiment.

3) @ notices changes taking place in an experiment and notes in
   particular those that are important to the investigation. As a result s/he
   generally has a sound grasp of the experiment.

4) @ is aware of changes taking place in an experiment and is aware
   of those which require measurement. This means s/he always has a
   thorough understanding of the ideas involved.

            Enter choice or use RETURN to enter your comment
     Use TAB to skip set, or 0 if you wish to add no more comments
            If you wish to enter more than one press R
```

A screen from the pupil report application written using InterBase

As would be expected, the main difference between InterBase and Basic is the addition of a whole host of database commands. As these are often self explanatory, they are easy to remember and for anyone with a rudimentary knowledge of Basic, learning can proceed at speed. READ REC (rec), GO START, USE DB <file> could hardly be more clear. The example database in the manual and on the disc is well documented and enables the user to make a start in constructing his or her system.

What has impressed me most about InterBase is its text handling facilities. The example which follows will hopefully serve to show its astonishing versatility.

Schools need to produce pupil reports which comment on achievement in a series of areas of learning. One way to do this is to use comment banks, selecting the most appropriate statement for each area, and compiling them into a report. In InterBase, two databases can be created. One contains basic pupil data, such as name, gender, form teacher and subject teacher (this incidentally can be imported from another database such as ViewStore, using an InterBase program provided on the disc). Each field, or a combination of fields can have its own index. For example, 160 pupils need to be sorted on the basis of teacher code, then pupil name. The second database contains the comment bank, with each record relating to a particular skill or area of knowledge, and each field relating to a level of achievement.

A number of commercial packages perform just this operation, but what is lacking is the facility for the user to word process each of the statements, or indeed the entire report. InterBase on the other hand has its own inbuilt text editor, invoked by the key word EDIT <string>. Many of the key controls are the same as in InterWord, so the user can treat the statements or the full report as if it were text in InterWord. Not only can the editor be used to edit records, but the database application programs themselves are written using it. A further use is in the production of 'Help' files. At any point in the execution of a program, the user can be given an opportunity to load a file containing a help message into the editor, or it can be used to make notes. In fact full instructions on the use of the application can be given in this file.

The editing function can also be invoked by the database application program itself. Automatic search and replace routines can be easily written, for example, to replace a chosen character with a name, or replace the string "s/he" with "he" or "she" depending on the gender indicated by the pupil file.

In most databases field lengths are limited to 255 characters. Not so in InterBase. Using the example program described, reports have been written which were close to one sheet of A4 in length, and the text could be edited in its entirety. Such a string is known as a 'Long String' in InterBase, and although a certain number of facilities are only available to 'Short strings' (255 characters or less), this does not normally pose a problem.

Once a program has been written in InterBase, it can be tokenised and installed as a ROM image in sideways RAM, leaving plenty of space for storage in main memory.

So far I have described a powerful database language in its own right. But as a member of the Inter family, its ability to link in memory with others of the family, (via the ROM link mechanism) opens up a world of opportunities. Again I need to turn to the application for report writing. Our pupil database contains pupil data together with the full report. It needs to be written in the context of the course the pupil has been studying. Details can be written in an InterWord file, and simple embedded commands used to extract the data from the database at the relevant points. Indeed, programs can be run from InterWord, so in the above example the database can be searched for an individual pupil while still in Interword. Even the name of the teacher can be inserted in place of the code which is held in the database. This facility makes use of the database header which in this case is used to contain the codes and the full name of the teacher.

If you are prepared to do a little programming, and already use the Inter series, then InterBase is for you. If you need to use any amount of text in your database, then there is no alternative.

InterBase arrived too late on the scene to get the attention it deserves. Without the attention of third party software writers, it has been seen by many only as a card index system, on the basis of the example supplied with it. Yet there are many thousands of BBCs in schools all over the country, and there is still a market for good software. We need a good database for pupil use, school record packages and so on, and InterBase is an ideal language.

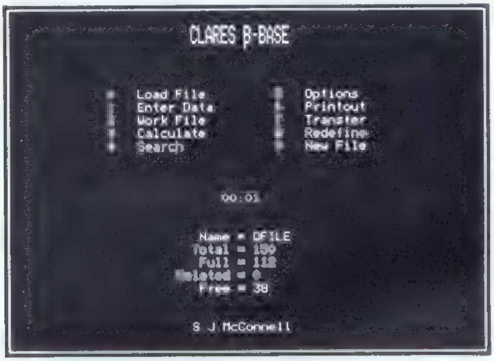
Beta-Base

"An oldie but a goodie" is Dave Futcher's assessment of Clares' long-standing database.

Beta-Base (Clares) £25.00 inc. VAT

During the last seven years BBC owners have had a considerable array of data handling packages available to them. Many of these have come and gone, but amazingly, one that I first used in 1983 and frequently still turn to, is still available.

Clares' *Beta-Base* (which later proved to be a forerunner of one of the first Archimedes database packages, *Alpha-Base*) first appeared in the early months of 1983 at a cost of £25.00. It immediately became popular, and by 1984 Clares had released an enhanced version of Beta-Base which was available to original users as an upgrade. This upgrade ensured that Beta-Base offered everything that its growing list of competitors did.



```
00:01

  Name = DFILE
 Total = 150
  Full = 112
Deleted = 0
  Free = 38

S J McConnell
```

The Beta-Base main menu, showing the options as described in the text

Despite its age Beta-Base has rightly stood the test of time. Many users who need a powerful disc-based random access database consider it to be first class. It is the ideal companion for those people who don't want to use ROM based packages.

But what are its main features? For a start, Beta-Base allows you to have plenty of fields, from 1 to 200 to be precise. If ten fields are specified, there is room for about 500 records on a 40 track disc. For names and addresses, it can easily manage 1200 records on the one disc.

Field size is reasonable, with up to 254 characters available, and the total record length can be up to 2048 characters.

The program supports full search and sort facilities. You can search 500 records on five fields in 60 seconds, and sort the same number on three fields in the same time. In addition, Beta-Base can handle calculations, and provides comprehensive print-out facilities.

Few computer users have time for applications that take a lot of understanding and learning. Beta-Base is certainly easy to use, however, and poses no problems in that quarter. Everything is controlled from menus, and the main menu options are called by pressing the number keys, as shown in the accompanying illustration. These options give access to all the main parts of the package.

Files are created from scratch with the *Enter Data* option and it is here that the record fields are first defined. Data entry can often be repetitive, so Beta-Base provides a *Global Entry* feature to enter whole records or any number of fields without having to re-type common data.

Browsing through a database is as simple as it could be, with good use of the cursor keys coupled with Shift. Beta-Base also has simple built-in editing features. 'G' enables you to go to a particular numbered record, 'E' lets you edit the record on screen, while 'D' will remove the record. I have always found the command 'P' (for printing the currently displayed record) invaluable.

When it comes to standard database tasks you will find that Beta-Base can naturally sort numerically and alphabetically in both ascending or descending order. But it also has some additional features often missing or difficult to use in other programs.

A very important feature is the *Search List*. This is a group of record numbers that have been found during a search, and once you make a Search List it is retained in memory until you either make another or leave the program. Usefully, a Search List can be saved and loaded thus allowing you to keep a list of all records that meet certain specific parameters. As the Search List is just an index of the records it is a very economical tool. It can also be used to hold records for sorting.

Functions such as calculate, print, transfer etc., can be used on all records or just a Search List, thus giving you the facility to perform these functions on selected records which meet your criteria. The Search List is a powerful concept and its use enables you to manipulate your data in many different ways.

As you can see, Beta-Base is well endowed with features that enable you to store, search and sort information. But it also offers calculations on numeric fields. These calculations can be performed globally across a number of records or locally by adding two fields together. The results of the calculation can then be stored in a nominated field. The array F(0) to F(n) is used for the calculations, with n equal to the number of fields in a record. Any valid expression can be used for the calculation, for example:

```
      F(5)=F(5)*1.15
```

Beta-Base has a powerful print option with a range of formats. You can print out records either from a Search List or between two specified positions in a file. Four main formats are available. The standard one is a record style print-out that emulates the screen display. Each record will be printed with a line for each field, and will include the record number and field titles.

Most useful is the non-standard format which offers a tabular print-out. This option provides a powerful means of controlling how you want the records printed. You can have the field titles on or off, you can change the order in which the fields are printed, or miss out some fields altogether. You can also print each record in a straight line and in columns if required. Two label styles are also available, with either one or two labels across the page.

Another powerful feature I've found useful in Beta-Base is its *Redefine* option. When creating a database you often find that after preparing and setting up the database you require an extra field, or you haven't allowed enough space within a string field to include all the information you want. You may even want to highlight a field by using teletext control codes. With Beta-Base all these things are possible thanks to this option.

Sometimes you also need to be able to transfer the whole or part of the contents of a file into another file. Not many database packages make this easy to accomplish, but Beta-Base does. The *Transfer* option will enable you to create new databases by taking parts from several others and merging them. The Search List can play a large part in the transfer option by allowing you to select the data for transfer.

I certainly rate Beta-Base as one of the best disc-based database packages for the BBC Micro. It may be an "oldie" but it remains a good all rounder - matching price with performance. It has all the features that a BBC database should have for the basic tasks that I need to perform.

Be warned, though, that it does show its age in some quarters! For instance having a field sort length of only ten characters can be problematic. Another drawback is that the program disc cannot be legally copied and therefore data has to be stored on a separate data disc.

Nevertheless, Beta-Base is well worth considering if you don't want a ROM-based package for data handling, and is still available at its 1983 price of £25.

SUMMARY

This concludes our survey of databases for the BBC micro. As with our previous survey on word processors, we welcome comments or hints from readers on this subject. We would also like to hear from you if you think that you may like to contribute to a future survey. We are considering a number of areas of BBC computing, and if you have worthwhile experience of a particular software package, and are confident of your ability to write about it, then please contact the editor.

ADDRESSES

Computer Concepts
Gaddesden Place, Hemel Hempstead, Herts HP2 6EX.
Tel. (0442) 63933.

Clares Micro Supplies
98 Middlewich Road, Northwich, Cheshire CW9 7DA.
Tel. (0606) 48511.

Wordprocessor Style Input (Part 1)

Andrew Rowland introduces a versatile and flexible editor for all forms of keyboard input.

All of us spend a lot of time at the keyboard entering single lines, whether it's typing and altering Basic programs, entering star commands or supplying input to programs. And we are all aware of the limitations of the BBC micro in this area - the only way to re-use a previous string is to copy it from the screen, and worst of all, you have to delete all the way back to mistakes.

What I would like is to be able to insert or delete characters at any point, recall the last line I entered, and do things like change the case of letters or swap two characters round, without losing features I am used to like screen copying. In other words, I want a replacement for the operating system's standard string input routine, Osword 0.

In this month's article I will present such a routine, called WPinput, which is suitable for the Model B and Master series. In the following two issues, we will put its powerful editing capabilities to further good use.

ENTERING THE PROGRAM

Ensure that PAGE is at &1900, type NEW and enter this month's listing. Type line 10 *exactly* as printed and omit lines 1020-1050 at first. There is no need to enter comment lines (preceded by '\'), but do keep line numbers as listed. Save the program as *Osword0* before testing it thoroughly. It alters operating system vectors, so any mistake could make the machine hang and require you to press Break. When all is well, enter lines 1020-1050 which keep the routine active even after pressing Ctrl-Break.

USING WPINPUT

When you run the program, you are asked if you want to save or install the routine. If saved, you can install it in future by typing *WPinput, but make sure PAGE is at or above &1900. This can be protected across Break by typing *FX180,25. Now the full power of the WPinput is at your fingertips! The routine will be used automatically at Basic's '>' prompt, by the keyword INPUT, at the View family's '=>' prompt, for entering star commands from Wordwise and the Master's Edit, and on many other occasions.

The cursor keys are used on three levels: on their own, for moving the cursor in a string, as in a word processor; with Shift for faster movement; and with Ctrl for cursor copying. The latter works in exactly the same way as before, except you must hold down Ctrl to move the flashing cursor to the part of the screen you want to copy from. When you do this, a blob cursor appears which is transparent in all but mode 7, and which can still be moved with the cursor keys alone. Copy is used as normal, or together with Shift to change the case of any letters being copied. Tab cancels cursor copying.

When Shift is used with the left or right cursor keys, the cursor moves a word at a time - a word being any group of characters bounded by a space or colon, so it treats Basic statements as words. Multiple spaces are treated as one word too, so it is always easy to get to the start or end of a word. Shift with the up or down cursor keys takes you to the beginning or end of the string.

You are no doubt familiar with Ctrl-U to delete a whole line. As well as this, Ctrl-D deletes the character above the cursor and Ctrl-X changes its case - Wordwise users may wish to change this to Ctrl-A and Ctrl-S respectively (lines 1980 and 1830), but should note that they would no longer be able to change screen colours or send codes to the printer using the Ctrl key. View users can achieve equal familiarity by typing *KEY 9 |D. Ctrl-Z transposes the current character with the one before it, and Ctrl-R recalls the last string entered, overwriting anything you may have just typed. There is a default string, recallable the first time you use WPinput in a session, which may be altered to something more useful (line 4320).

MEMORY USAGE

The code occupies 1.5K, plus a page to store the last string for the recall feature. If you use a Model B with DFS, it will just fit in the DFS workspace from &1300 without raising PAGE. Note that Model B owners should not change filing systems, type *DISC or open more than one file. Master owners must put up with the raised value of PAGE until next month, when I will show how you can get round these limitations by running it in sideways RAM. I will also explore the use of the routine in more depth then and give more technical detail.

```
   10 REM Program .>Osword0
   20 REM Version 1.10
   30 REM Machine BBC B/B+
   40 REM Author  Andrew Rowland
   50 REM BEEBUG  November 1990
   60 REM Program subject to copyright
   70 :
  100 IF PAGE<>&1900 THEN P%=PAGE+16:PAG
E=&1900:CHAIN $P%
  110 mc%=&1300:*FX180,25
  120 READ buffer,maxlen,minchar,maxchar
  130 DATA &E8,&2B3,&2B4,&2B5
  140 READ length,xtemp,ytemp,cursor
  150 DATA &A8,&A9,&AA,&AB
  160 READ width,ysize,xpos,ypos
  170 DATA &AC,&AE,&AF,&F3
  180 READ xsep,ysep,chbuff,temp,shftfl
  190 DATA &364,&365,&70,&F0,&F2
  200 oswrch=&FFEE:osbyte=&FFF4
  210 osrdch=&FFE0:osnewl=&FFE7
  220 osword=&FFF1:NETV=&224
  230 FOR pass=0 TO 2 STEP 2
  240 P%=mc%
  250 PROCass:NEXT:PROCend
  260 END
  270 :
 1000 DEF PROCass
 1010 [OPT pass
 1020 .install
 1030 LDA #&4C:STA &287
 1040 LDA #break MOD 256:STA &288
 1050 LDA #break DIV 256:STA &289
 1060 .vectors \ alter vectors
 1070 LDA &20C:CMP #entry MOD &100
 1080 BEQ insto \ already altered!
 1090 STA oldv:LDA &20D:STA oldv+1
 1100 SEI:LDA #entry MOD &100:STA &20C
 1110 LDA #entry DIV &100:STA &20D
 1120 .insto CLI:RTS
 1130 :
 1140 .break BCS vectors:RTS
 1150 \ ******************************
 1160 .oldv   EQUW 0
 1170 .status EQUB 0
 1180 .flag   EQUB 0
 1190 .xtable \ screen width per mode
 1200 EQUB 80:EQUB 40:EQUB 20:EQUB 80
 1210 EQUB 40:EQUB 20:EQUB 40:EQUB 40
 1220 .ytable \ no. lines
 1230 EQUB 32:EQUB 32:EQUB 32:EQUB 25
 1240 EQUB 32:EQUB 32:EQUB 25:EQUB 25
 1250 \ ******************************
 1260 .entry CMP #0:BEQ transfer
 1270 JMP (oldv)
 1280 .transfer
 1290 STA length:STX temp:STY temp+1
 1300 LDA &256:BEQ transok
 1310 LDA length:JMP (oldv)
 1320 .transok LDY #4
 1330 .translp
 1340 LDA (temp),Y:STA maxlen-2,Y
 1350 DEY:CPY #2:BCS translp
 1360 LDA (temp),Y:STA buffer+1
 1370 DEY:LDA (temp),Y:STA buffer
 1380 STY cursor:STY flag
 1390 LDX &27D:STX status
 1400 LDA #4:LDX #1:LDY #&FF:JSR osbyte
 1410 LDA #&50:ORA &27C:STA &27C
 1420 LDA #134:JSR osbyte
 1430 STX xpos:STY ypos
 1440 LDY &355:LDA xtable,Y:STA width
 1450 LDA ytable,Y:STA ysize
 1460 LDA #8:BIT &D0:BEQ nowindow
 1470 LDA &30A:SEC:SBC &308
 1480 STA width:INC width
 1490 LDA &309:SEC:SBC &30B
 1500 STA ysize:INC ysize
 1510 .nowindow LDY #8
 1520 .pushlp LDA chbuff,Y:PHA
 1530 DEY:BPL pushlp
 1540 .prstring JSR curoff:JSR display
 1550 JSR chkscroll:JSR calccursor
 1560 .movcur JSR prcursors:JSR curon
 1570 .getch LDA #0:STA shftfl
 1580 JSR osrdch:BCC notesc:JMP exit
 1590 .notesc TAX
 1600 LDA &27C:ROL A:ROL A:BCS over
 1610 LDA &26A:BEQ over
 1620 TXA:JSR oswrch:JMP getch
 1630 .over TXA:CMP #&87:BCC notcurkey
 1640 CMP #&8C:BCS notcurkey
 1650 JMP curskeys
 1660 .notcurkey CMP #127:BNE notdel
 1670 JSR delete:JMP prstring
 1680 :
 1690 .compare CMP minchar:BCC notin
 1700 CMP maxchar:BEQ addit:BCC addit
 1710 .notin JMP peep
 1720 .addit JSR insert:JMP prstring
 1730 :
 1740 .notdel CMP #32:BCS compare
 1750 CMP #12:BNE notCLS
 1760 LDX #0:STX xpos:STX ypos
 1770 .notCLS CMP #ASC"U"-&40:BNE notU
 1780 JSR delline:BEQ prstring
 1790 .notU
 1800 CMP #9:BNE nottab:LDA #0:STA flag
 1810 BEQ prstring
 1820 .nottab
 1830 CMP #ASC"X"-&40:BNE notX
 1840 LDY cursor:CPY length
 1850 BCC Xok:JMP peep
 1860 .Xok LDA (buffer),Y:JSR case
 1870 STA (buffer),Y:INC cursor
 1880 JMP prstring
 1890 .notX CMP #ASC"Z"-&40:BNE notZ
 1900 LDY cursor:CPY length
 1910 BCC Zok1:JMP peep
 1920 .Zok1 CPY #1:BCS Zok2:JMP peep
 1930 .Zok2 LDA (buffer),Y:PHA:DEY
 1940 LDA (buffer),Y:INY:STA (buffer),Y
 1950 DEY:PLA:STA (buffer),Y:JMP prstrin
g
 1960 .notZ CMP #ASC"R"-&40:BNE notR
 1970 JSR recall:JMP prstring
 1980 .notR CMP #ASC"D"-&40:BNE notD
 1990 LDX cursor:CPX length
 2000 BCC Dok:JMP peep
 2010 .Dok INC cursor:JSR delete
 2020 JMP prstring
 2030 .notD CMP #13:BEQ out
 2040 JSR oswrch:JMP getch
 2050 :
 2060 .out LDY length:STA (buffer),Y
 2070 JSR tellnet
 2080 .keep CPY #0:BEQ exit:STY store
 2090 .kloop
 2100 LDA (buffer),Y:STA store+1,Y
 2110 DEY:BNE kloop
 2120 LDA (buffer),Y:STA store+1,Y
 2130 .exit LDX xpos:LDY ypos:JSR tab
 2140 LDA #&AF:AND &27C:STA &27C
 2150 JSR showstring:JSR osnewl
 2160 LDA #4:LDX status:JSR osbyte
 2170 LDY #0
 2180 .pullp PLA:STA chbuff,Y
 2190 INY:CPY #9:BNE pullp
 2200 LDY length:LDA &FF:ASL A
 2210 LDA #0:TAX:RTS
 2220 :
 2230 .curskeys PHA:LDA #129:LDY #&FF
 2240 LDX #-2 AND &FF:JSR osbyte
 2250 CPX #&FF:BEQ ctrlpressed
 2260 LDA #129:LDY #&FF
 2270 LDX #-1 AND &FF:JSR osbyte
 2280 CPX #&FF:BNE neither
 2290 JMP shiftpressed
 2300 .neither PLA
 2310 CMP #&87:BNE left:JMP copy
 2320 .left CMP #&88:BNE right
 2330 LDA cursor:BEQ peep
 2340 DEC cursor:JMP currout
 2350 .right CMP #&89:BNE down
 2360 LDA cursor:CMP length:BEQ peep
 2370 INC cursor:BNE currout
 2380 .down CMP #&8A:BNE up
 2390 LDA cursor:CLC:ADC width
 2400 BCS peep:CMP length
 2410 BEQ currok:BCS peep:BCC currok
 2420 .up LDA cursor
 2430 SEC:SBC width:BCC peep
 2440 .currok STA cursor
 2450 .currout JMP prstring
 2460 :
 2470 .peep LDA #7:JSR oswrch:JMP getch
 2480 :
 2490 .ctrlpressed PLA:CMP#&87:BEQ copy
 2500 .chkfirst LDX flag:BNE notfirst
 2510 INC flag:LDX xtemp:STX xsep
 2520 LDX ytemp:STX ysep
 2530 .notfirst CMP #&88:BNE ctrlri
 2540 JSR decx:JMP send
 2550 .ctrlri CMP #&89:BNE ctrlup
 2560 JSR incx:JMP send
 2570 .ctrlup CMP #&8B:BNE ctrldown
 2580 JSR decy:JMP send
 2590 .ctrldown JSR incy
 2600 .send JMP movcur
 2610 :
 2620 .shiftpressed INC shftfl
 2630 PLA:CMP #&8B:BEQ top
 2640 CMP #&8A:BEQ bottom
 2650 CMP #&88:BEQ wordleft
 2660 CMP #&89:BEQ wordright
 2670 :
 2680 .copy LDA flag:BEQ peep
 2690 .copok LDA #135:JSR osbyte
 2700 TXA:BEQ peep
 2710 LDX shftfl:BEQ copins:JSR case
 2720 .copins JSR incx:JMP compare
 2730 :
 2740 .top LDA #0:STA cursor
 2750 JMP prstring
 2760 :
 2770 .bottom LDA length:STA cursor
 2780 JMP prstring
 2790 :
 2800 .wordleft JSR moveleft
 2810 BCC wlov1:JMP peep
 2820 .wlov1 BNE wlletter
 2830 .wlsploop JSR moveleft
 2840 BCS wlout:BEQ wlsploop
 2850 JSR moveright
 2860 .wlout JMP prstring
 2870 .wlletter JSR moveleft
 2880 BCS wlout:BNE wlletter:BCC wlout
 2890 :
 2900 .moveleft LDY cursor:BEQ mlnotok
 2910 DEY:CPY #0:BCS mlok
 2920 .mlnotok SEC:RTS
 2930 .mlok STY cursor:LDA (buffer),Y
 2940 CMP #32:BEQ mlo:CMP #ASC":"
 2950 .mlo CLC:RTS
 2960 :
 2970 .wordright JSR moveright
 2980 BCC wrov1:JMP peep
 2990 .wrov1 BNE wrletter
 3000 .wrsploop JSR moveright
 3010 BCS wrout:BEQ wrsploop
 3020 .wrout JMP prstring
 3030 .wrletter JSR moveright
 3040 BCS wrout:BNE wrletter
 3050 JSR moveright:PHP:DEC cursor
 3060 PLP:BCS wrout:BEQ wrout
 3070 JSR moveright:JMP prstring
 3080 :
 3090 .moveright LDY cursor:INY
 3100 CPY length:BEQ mlok:BCC mlok:RTS
 3110 :
 3120 .tellnet JMP (NETV)
 3130 :
 3140 .delline LDX length:INX
 3150 STX cursor:JSR calccursor
 3160 LDX xtemp:LDY ytemp:JSR tab
 3170 LDA #127:LDX length:BEQ delliout
 3180 .dellilp JSR oswrch
 3190 DEX:BNE dellilp
```

```
3200 .delliout STX length:STX cursor
3210 RTS
3220 :
3230 .display LDA #0:STA &269
3240 LDX xpos:LDY ypos:JSR tab
3250 .showstring
3260 LDX length:BEQ dispout:LDY #0
3270 .displp LDA (buffer),Y:JSR oswrch
3280 INY:DEX:BNE displp
3290 .dispout
3300 LDA #32:JSR oswrch:JMP oswrch
3310 :
3320 .chkscroll LDA cursor:PHA
3330 LDX length:INX:INX:STX cursor
3340 .chklp JSR calccursor
3350 CMP ysize:BCC scrollout
3360 DEC ypos:JSR decy:JMP chklp
3370 .scrollout PLA:STA cursor:RTS
3380 :
3390 .insert LDY length:CPY maxlen
3400 BNE insover
3410 .peepout LDA #7:JMP oswrch
3420 .insover PHA:JSR pad:PLA
3430 LDY cursor
3440 .insout STA (buffer),Y
3450 INC cursor:INC length:RTS
3460 :
3470 .pad LDY length:TYA:DEY:SEC
3480 SBC cursor:BEQ padout:TAX
3490 .padlp
3500 LDA (buffer),Y:INY:STA (buffer),Y
3510 DEY:DEY:DEX:BNE padlp
3520 .padout RTS
3530 :
3540 .delete LDY cursor:BEQ peepout
3550 .dellp
3560 LDA (buffer),Y:DEY:STA (buffer),Y
3570 INY:INY:CPY length:BCC dellp
3580 .delout DEC cursor:DEC length:RTS
3590 :
3600 .calccursor LDA cursor
3610 CLC:ADC xpos:LDX #0
3620 .calclp STX ytemp:INX:STA xtemp
3630 SEC:SBC width:BCS calclp
3640 LDA ytemp:CLC:ADC ypos
3650 STA ytemp:RTS
3660 :
3670 .prcursors
3680 LDX xtemp:LDY ytemp:JSR tab
3690 LDA flag:BEQ notsep:JSR reverse
3700 LDX xsep:LDY ysep:JSR tab
3710 .notsep RTS
3720 :
3730 .reverse LDA #32:LDY cursor
3740 CPY length:BEQ revov
3750 LDA (buffer),Y
3760 .revov STA chbuff:LDA #10
3770 LDX #chbuff:LDY #0:JSR osword
3780 LDA #23:JSR oswrch
3790 LDA #255:JSR oswrch
3800 LDX #0:LDY #8
3810 .revlp LDA #&FF:EOR chbuff+1,X
3820 JSR oswrch:INX:DEY:BNE revlp
3830 LDA #255:JMP oswrch
3840 :
3850 .recall LDX store:BEQ rexit
3860 JSR delline
3870 LDY #0:LDX #0:STY cursor
3880 .rloop
3890 LDA store+1,X:CMP maxchar:BEQ rok
3900 BCS rlov:CMP minchar:BCC rlov
3910 .rok STA (buffer),Y:INY
3920 CPY maxlen:BCS rout
3930 .rlov INX:CPX store:BNE rloop
3940 .rout STY length
3950 .rexit RTS
3960 :
3970 .incx INC xsep:LDX xsep:CPX width
3980 BCC inxo:LDX #0:STX xsep:JSR incy
3990 .inxo RTS
4000 :
4010 .decx DEC xsep:BPL dexo
4020 LDX width:DEX:STX xsep:JSR decy
4030 .dexo RTS
4040 :
4050 .incy INC ysep:LDX ysep:CPX ysize
4060 BNE inyo:LDX #0:STX ysep
4070 .inyo RTS
4080 :
4090 .decy DEC ysep:BPL deyo
4100 LDX ysize:DEX:STX ysep
4110 .deyo RTS
4120 :
4130 .tab LDA #31:JSR oswrch
4140 TXA:JSR oswrch
4150 TYA:JMP oswrch
4160 :
4170 .case TAX:AND #&DF
4180 CMP #65:BCC notletter
4190 CMP #91:BCS notletter
4200 TXA:EOR #&20:RTS
4210 .notletter TXA:RTS
4220 :
4230 .curon  LDX #1:BNE cur
4240 .curoff LDX #0
4250 .cur    LDA #23:JSR oswrch
4260 LDA #1:JSR oswrch:TXA:JSR oswrch
4270 LDA #0:LDY #7
4280 .curlp
4290 JSR oswrch:DEY:BPL curlp:RTS
4300 :
4310 .store EQUB LEN($(store+1))
4320 EQUS "Coded by Andrew Rowland":EQU
B 13
4330 ]ENDPROC
4340 :
4350 DEF PROCend:REPEAT
4360 PRINT'"Install, save or quit? [I/S
/Q]":*FX21
4370 REPEATM$=CHR$(GET AND &DF):UNTILIN
STR("ISQ",M$)
4380 IFM$="I" CALL install
4390 IFM$="S" a$="SAVE WPinput "+STR$~m
c%+" "+STR$~P%
4400 IFM$="S" PRINT a$:OSCLI a$
4410 UNTILM$="Q":ENDPROC
```

Searching (Part 3)

by Bernard Hill

This month we continue our searching theme with a more mathematical flavour. We are going to consider the search for a solution to an equation. We shall be looking at a simple algorithm to demonstrate the principles, but there are other more complex methods available which will find a solution more quickly. Consider the following equation:

x - SIN(x) = PI/2

This gives the size of a segment of a circle which is a quarter of the area of the circle where x is the angle in radians subtended from the centre.

Now this is no ordinary equation which can be cracked by elementary algebra. Surprisingly enough, one of the standard ways of solving it is by guesswork. Suppose we try x=0 in the equation. The left-hand side is then 0, but if we put x=PI then the left-hand side becomes PI and that's too big. So there's a solution somewhere between 0 and PI. In fact we can set this out as in table1.

Note that I have subtracted PI/2 from both sides so that we are trying to find the x which makes the expression zero.

We can see that a solution for x lies between 2.2 and 2.4. So we could start with x at 2.2 and increase it by (say) 0.01 to produce another table and find a pair of values where the function changes sign (these would be 2.30 and 2.31), and carry on decreasing the search increment by 10 each time to progressively approach the correct answer. Here is a program to solve this equation to 6 decimal places:

```
  10 REM search from 0 to 3 steps of 0.1
  20 PROCsolve(0,3,0.1,1E-6)
  30 END
  40:
1000  DEF PROCsolve(left,right,step,acc)
1010 LOCAL x,sgn,sgn2
1020 IF step<acc THEN PRINT "Solution is";
     (left+right)/2:ENDPROC
1030 x=left:sgn=SGN(FNf(x))
1040 REPEAT
1050 x=x+step:sgn2=SGN(FNf(x))
1060 UNTIL x>=right OR sgn<>sgn2
1070  IF sgn=sgn2 THEN PRINT "No solution"
:END
1080 PROCsolve(x-step,x,step/10,acc)
1090 ENDPROC
1100:
2000 DEF FN(x)=x-SIN(x)-PI/2
```

x	x - SIN(x) - PI/2
0.0	-1.5708
0.2	-1.5695
0.4	-1.5602
0.6	-1.5354
0.8	-1.4882
1.0	-1.4123
1.2	-1.3028
1.4	-1.1562
1.6	-0.9704
1.8	-0.7446
2.0	-0.4801
2.2	-0.1793
2.4	0.1537
2.6	0.5137
2.8	0.8942
3.0	1.2881

Table 1

Note that the program is recursive and we stop when the step length is within the limit of accuracy required, quoting the middle of the range at that point as the solution.

Now we can make a number of improvements on PROCsolve:

1. As given above the program stops when it finds the first solution: we can make it find all solutions within the range by placing the recursive call to PROCsolve within the REPEAT loop.

2. It only works for FNf(x). We can make it more general by replacing FNf(x) in lines 1030 and 1050 with EVAL(f$), where f$ is another parameter which would evaluate to the function:

```
 20 PROCsolve("x-SIN(x)-PI/2",0,3,0.1,1E-6)
```

and:

```
1000 DEF PROC(f$,left,right,step,acc)
```

There is also the question of the "/10" in line 1080. We only divided by 10 on each finer search because we are human and think in decimal. If we replace the 10 by 2 then we have what is known as the "bisection method" of solving the equation. This requires far fewer evaluations of the function than with a 10, resulting in a greater speed for complicated functions. The drawback for us is that it requires a deeper level of recursion (involving time and space) and we might even hit the limit of the REPEAT loop during recursion (see last month's article). In fact a good compromise on the Beeb seems to be about 4.

WEAKNESSES IN THE ALGORITHM

If we try to automate the program to solve an arbitrary equation then we shall run into difficulties. It may be that the initial search interval is too large, and that between (say) two positive values in the search, we have missed two solutions where the values went below 0 and quickly back. Now provided we know we've done this then we can always search in a difficult region with a finer grid, but if we try to automate the problem we have to write a very intelligent program!

Another problem comes in trying to solve the equation:

  1/(x-PI)=0

with:

```
20 PROCsolve("1/(x-PI)",0,10,1,1E-6)
```

Running this tells us that we have a solution of x=3.141593 - and that's precisely where the formula is infinite, not zero! What we have actually written is a program to find out where the function changes sign, not where it is zero - not quite the same thing at all, and we'll need to watch out for spurious solutions such as this.

Equation-solving by computer is fraught with these and similar problems - there is no one algorithm which will solve every equation, and it's usually a very good idea to have a graph of the function to hand so that you have an idea where it crosses the axes and how often. Fortunately it doesn't take much to flesh the PROCedure above into a fully-fledged equation-solving program with graphical interaction and a menu interface. This is the program below and I leave you to explore all sorts of equations with it.

OPERATION OF THE PROGRAM

The program is pretty self-explanatory, but note the following points:

1. The Function field should be entered with a function of lower-case x such as:

  x-SIN(x)-PI/2.

2. The equation to be solved is Function=0, so bring any constants over to the left-hand side. To solve:

  x^x=2

give:

  x^x-2

as the function.

3. Any legal Basic expression can be given as the equation, and any errors in the expression will cause the program to stop when plotting and report the error. This also applies to run-time errors such as '-ve root' or 'Too big'.

4. The y-scale field is given so that you can see more or less of the y values (larger y-scale values give greater magnification).

5. Changing the x range will cause the step to change to give 500 x-points on the screen, but you can override the step length if you want.

6. Pressing Escape while plotting will move back to the menu screen quoting the last solution found. Pressing Escape from the menu screen stops the program.

7. If obviously bad data is given, then the program refuses to plot and the cursor is placed over the bad field (e.g. step=0, or a function without an 'x').

8. You'll have to be intelligent about the solutions reported: watch out for the infinities as well as the zeros as illustrated above.

Happy solving!

```
   10 REM EQUATION SOLVER
   20 REM Version 1.0
   30 REM Author  Bernard Hill
   40 REM Beebug  November 90
   50 REM Program subject to copyright
   60 :
   70 REM pretty random start values...
```

```
  100 xmin=0:xmax=100:f$="SINx-0.02*x"
  110 st=(xmax-xmin)/500:acc=1.E-6
  120 yscale=200
  130 DIM xp(5),yp(5),P$(5),L(5)
  140 FOR i=0 TO 5
  150 READ yp(i),P$(i)
  160 xp(i)=LENP$(i)+6:L(i)=34-LENP$(i)
  170 NEXT:g$=CHR$130:y$=CHR$131
  180 b$=CHR$132+CHR$157+CHR$131
  190 e$="  "+CHR$156:L(2)=L(2)+39
  200 T$=b$+CHR$141+"     BEEBUG Equatio
n Solver"
  210 *fx4,1
  220 ON ERROR GOTO 4120
  230 error=FALSE:N%=0
  240 MODE7:plotting=FALSE:PRINT T$'T$
  250 FOR i=0 TO 5
  260 PRINTTAB(0,yp(i))b$;P$(i);e$
  270 NEXT
  280 PRINTTAB(0,18)b$"Last solution  "e
$;
  290 IF N%=0 THEN PRINT"NONE" ELSE PRIN
Troot
  300 @%=&90A
  310 PRINTTAB(0,20)b$"Solutions found"e
$;N%
  320 PRINTTAB(10,22)"Press COPY to star
t"
  330 c=0:REPEAT:REPEAT
  340 PRINTTAB(xp(0),yp(0));xmin
  350 PRINTTAB(xp(1),yp(1));xmax
  360 PRINTTAB(xp(2),yp(2));f$
  370 PRINTTAB(xp(3),yp(3));st
  380 PRINTTAB(xp(4),yp(4));acc
  390 PRINTTAB(xp(5),yp(5));yscale
  400 IF error THEN PRINTTAB(0,22);CHR$1
36"An error has occurred, check Function
";:REPORT
  410 VDU31,xp(c),yp(c):n=1:a$=""
  420 REPEAT x=GET
  430 IF n=1 THEN IF FNend(x) THEN 550
  440 IF n=1 THEN PRINTSTRING$(L(c)," ")
;TAB(xp(c),yp(c));
  450 IF x=127 THEN IF n>1 THEN VDU8,32,
8:a$=LEFT$(a$,LENa$-1):n=n-1::GOTO 550
  460 IF x=127 THEN VDU7:GOTO 550
  470 IF NOT FNend(x) THEN IF n<=L(c) TH
EN a$=a$+CHR$x:VDUx:n=n+1:GOTO 550
  480 IF c=0 THEN xmin=VALa$:a$=STR$xmin
  490 IF c=1 THEN xmax=VALa$:a$=STR$xmax
  500 IF c=2 THEN f$=a$
  510 IF c=3 THEN st=VALa$:a$=STR$st
  520 IF c=4 THEN acc=VALa$:a$=STR$acc
  530 IF c=5 yscale=VALa$:a$=STR$yscale
  540 IF c<2 THEN st=(xmax-xmin)/500:PRI
NTTAB(xp(3),yp(3));st;SPC20
  550 UNTIL FNend(x)
  560 PRINTTAB(xp(c),yp(c));STRING$(L(c)
," ")
  570 IF x=13 OR x=138 THEN c=(c+1)MOD6
  580 IF x=139 THEN c=(c+5) MOD 6
  590 UNTIL x=135
  600 bad=-1:IF st<=0 THEN bad=3
  610 IF xmax<xmin THEN bad=0
  620 IF yscale<=0 THEN bad=5
  630 IF INSTR(f$,"x")=0 THEN bad=2
  640 IF bad>-1 THEN VDU7,7:c=bad
  650 UNTIL bad=-1
  660 MODE0:plotting=TRUE:N%=0
  670 PROCshowsolve(f$,xmin,xmax,st,acc,
yscale)
  680 REPEAT UNTIL GET=32
  690 GOTO 240
  700 :
 1000 DATA 3,"Min x   "
 1010 DATA 5,"Max x   "
 1020 DATA 7,"Function"
 1030 DATA 9,"Step    "
 1040 DATA 11,"Accuracy"
 1050 DATA 13,"Y-Scale "
 1060 :
 2000 DEF FNend(x)=(x=13 OR x>128)
 2010 :
 3000 DEF PROCshowsolve(f$,left,right,st
ep,acc,yscale)
 3010 PRINTTAB(0,30)"X range "left" to "
right;TAB(40,30);"Y range ";-512/yscale;
" to ";512/yscale
 3020 @%=&20000+256*INT(-LOGacc)
 3030 xscale=1280/(right-left)
 3040 MOVE 0,512:DRAW 1279,512
 3041 VDU29,0;512;
 3042 PROCsolve(f$,left,right,step,acc)
 3070 ENDPROC
 3080 :
 4000 DEF PROCsolve(f$,a,b,step,acc)
 4010 LOCAL x,y,sgn,sgn2
 4020 IF step<acc THEN N%=N%+1:root=(a+b
)/2:SOUND0,-9,50,2:PRINTTAB(0,0)"Last so
lution:"root;SPC30:ENDPROC
 4030 x=a:y=EVALf$:sgn=SGNy
 4040 PLOT 69,xscale*(x-xmin),y*yscale
 4050 REPEAT x=x+step:y=EVALf$
 4060 PLOT 69,xscale*(x-xmin),y*yscale
 4070 sgn2=SGNy
 4080 IF sgn2<>sgn THEN PROCsolve(f$,x-s
tep,x,step/4,acc):sgn=sgn2
 4090 UNTIL x>=b
 4100 ENDPROC
 4110 :
 4120 error=ERR<>17
 4130 IF plotting THEN 240
 4140 IF ERR=17 MODE7:OSCLI("fx4"):END
 4150 REPORT:PRINT" at line ";ERL:END
```

Phone Call Costing Goes International

Phil Skelton extends this popular program to cope with international calls, calls to the Irish Republic and even freephone calls and calls to mobile phones.

It was said by David Williams that his *Phone Call Costing* program, BEEBUG Vol.9 No.3 (July 1990), could be easily amended or updated to accommodate international calls. As it turned out, it was not as straightforward as one might have expected, taking some midnight oil to sort out the oddities. What we have now is an all-singing, all-dancing version that brings this program into a form that will check the cost of calls to overseas territories, to mobile phones, to the Irish Republic, and will also indicate a zero charge for freephone calls. To achieve this a fairly substantial update to the original program is needed, and this is given below. When using the revised program for international calls, enter '010' followed immediately by the country code in response to 'Dialling code'. Enter area code and actual number together in response to 'Enter phone number'.

Having relatives that live far away prompted this effort, as it is only too easy to end up with a horrendous phone bill, if you get carried away with a couple of calls to Singapore or Oz.

UPDATING THE ORIGINAL PROGRAM

To effect this update the lines described below will have to be changed or added. Remember that you can modify the dialling codes in all UK categories to suit your own circumstances (what constitutes a local call code depends on where you live - see the original article and program for more information on this).

Line numbers given below refer to the *original* program listing, but you are advised to renumber the original program from the beginning in steps of 100 in order to make sufficient room for the additions. For this reason the additional coding is given without line numbers. The updates can also be applied equally to the model B or Master 128 version of the program.

Between 160 and 170 you should insert:

```
IF dist$="Error" G$="":GOTO190
```

change line 190 to read:

```
190 UNTIL G$=" " OR dist$="Error"
```

and add a new line between 190 and 200:

```
IF dist$="Error" GOTO250
```

Change the GOTO line numbers as appropriate. Between 1390 and 1400 insert:

```
TIME=0:REPEAT:UNTIL TIME=50
```

Lines 1670, 1680 and 1690 should be changed to read:

```
1670 IF len<1 OR len>8THEN flag=FALSE
1680IF LEN(code$)>10 THEN flag=FALSE
1690 IF LEN(code$)>18 THEN flag=FALSE
```

To overcome the problem with the use of STR$ in *message$*, lines 1980, 1990 and 2000 should be changed to read:

```
 1980 message$="Rate="+rate$+":Distance=
"+dist$+":Secs/unit=":len=LENmessage$
 1990spc%=19-(len+3) DIV2
 2000PRINTTAB(0,22)SPC(spc%)CHR$132;mess
age$;(unitsec)SPC2
```

Between 2260 and 2270 add an additional line:

```
 IF chargesec=0 AND unitsec=0 AND u
nitcharge=0 THEN cost%=0:ENDPROC
```

A total of 14 extra lines are needed between the original lines 2340 and 2350 for the International dialling codes as follows:

```
IF dist$="D" PROCrate1a:ENDPROC
IF dist$="E" PROCrate1b:ENDPROC
IF dist$="G" PROCrate1c:ENDPROC
                 IFday>5ANDdist$="A"
dist=6:PROCalpha:ENDPROC
IF day<6 AND dist$="A" PROCrate1a:ENDPROC
```

```
 IF day>5 AND dist$="A2" dist=7:PROCalpha:
ENDPROC
 IF day<6 AND dist$="A2" PROCrate1a:ENDPRO
C
 IF day>5 AND dist$="B" dist=8:PROCalpha:E
NDPROC
 IF day<6 AND dist$=B" PROCrate1a:ENDPROC
 IF day>5 AND dist$="C" dist=9:PROCalpha:E
NDPROC
 IF day<6 AND dist$="C" PROCrate1a:ENDPROC
 IF day>5 AND dist$="F" dist=12:PROCalpha:
ENDPROC
 IF day<6 AND dist$="F" PROCrate1a:ENDPROC
 IF dist$="ZERO" rate$="FREE":ENDPROC
```

Insert between lines 2510 and 2520:

```
 REPEAT READ local$
 IF local$=code$ dist=14:dist$="ZERO"
 UNTIL dist=14 OR local$="0000"
 IF dist=14 unitcharge=0:ENDPROC
 unitcharge=5.06
```

In lines 2540, 2580, 2610, 2680, 2790 and 2820, change "000" to "0000". Now comes the coding to detect the additional dialling codes. If in the future there are any changes in the number of zones, this is where it may be necessary to add or delete a routine. I have adhered to the zone letters as used by British Telecom for dist$. The following lines have to be inserted between lines 2830 and 2840:

```
 REPEAT READ local$
 IF local$=code$ dist=5:dist$="M"
 UNTIL dist=5 OR local$="0000"
 IF dist=5 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=6:dist$="A"
 UNTIL dist=6 OR local$="0000"
 IF dist=6 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=7:dist$="A2"
 UNTIL dist=7 OR local$="0000"
 IF dist=7 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=8:dist$="B"
 UNTIL dist=8 OR local$="0000"
 IF dist=8 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=9:dist$="C"
 UNTIL dist=9 OR local$="0000"
 IF dist=9 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=10:dist$="D"
 UNTIL dist=10 OR local$="0000"
 IF dist=10 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=11:dist$="E"
 UNTIL dist=11 OR local$="0000"
 IF dist=11 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=12:dist$="F"
 UNTIL dist=12 OR local$="0000"
 IF dist=12 ENDPROC
 REPEAT READ local$
 IF local$=code$ dist=13:dist$="G"
 UNTIL dist=13 OR local$="0000"
 IF dist=13 ENDPROC
 IF local$="0000"AND LEFT$(code$,3)= "010"
THEN PROCdialerror
```

Next comes the data for all of the additional dialling codes, with first the freephone data which goes between lines 2630 and 2640:

```
 REM data freephone calls
 DATA 0800,0073,0426,0000
```

and the rest of the data goes between lines 2820 and 2830:

```
 REM DATA 'M' RATE AND IRISH CALLS

 DATA 0001,010353,0860,08362,08363
 DATA 08364,08365,08366,08367,0898
 DATA 0033,0034,0035,0036,0037,0038
 DATA 0039,0000

 REM INTERNATIONAL CHARGE BAND 'A'

 DATA 01033628,01032,01045,010298
 DATA 01033,01049,0104175,010352
 DATA 0103393,01031,01041,0000

 REM INTERNATIONAL CHARGE BAND 'A2'

 DATA 010351,01034,010233,01039
 DATA 01035191,01039549,0000

 REM INTERNATIONAL CHARGE BAND 'B'

 DATA 01042,010357,01043,010358
 DATA 01030,01036,010356,01047
 DATA 01048,0000

 REM INTERNATIONAL CHARGE BAND 'C'

 DATA 010213,0101809497,010501
 DATA 0101809,0101809929,0101809923
 DATA 010359,010180994,0101809449
 DATA 010180944,01037,0101809440
```

```
DATA 010590,010509,010354,010218
DATA 010596,0101809491,010212,0101
DATA 0101809465,010180945,010508
DATA 010216,01090,0101809949,01038
DATA 0000

REM INTERNATIONAL CHARGE BAND 'D'

DATA 010559,010297,01053,010592
DATA 010965,010968,010974,01040
DATA 010966,01027,010971,0107,0000

REM INTERNATIONAL CHARGE BAND 'E'

DATA 01061,010852,01031,01065,0000

REM INTERNATIONAL CHARGE BAND 'F'

DATA 01054,010973,010229,010267
DATA 01055,010237,01056,010506
DATA 010255,01020,010503,010251
DATA 010241,010220,010223,010502
DATA 010504,01098,010964,010972
DATA 010962,010254,010961,010266
DATA 010231,010265,010230,010264
DATA 010505,010234,010507,010595
DATA 01051,010221,010248,010232
DATA 010252,010249,010597,010963
DATA 010255,010228,010256,010598
DATA 01058,010967,010260,010263
DATA 0000

REM INTERNATIONAL CHARGE BAND 'G'

DATA 010244,010247,010880,010591
DATA 010673,010266,01095,010257
DATA 010236,01086,010672,01057
DATA 010242,010682,010593,010500
DATA 010679,010594,010689,010299
DATA 010671,01091,01062,01081
DATA 010686,01082,010853,010261
DATA 01060,010690,010223,010962
DATA 010222,01052,010691,010258
DATA 010674,010977,0101809469
DATA 010687,010227,0106723,010670
DATA 01092,010675,01063,010262
DATA 010250,010682,010685,010677
DATA 01094,010886,01066,010675
DATA 010678,010969,010243,0000
```

That is all of the data for overseas calls at the moment. There are a number of territories that are not yet available to be dialled direct, but when and if they come on line it only needs the dialling code to be inserted in the appropriate data block. British Telecom will let you know when the time comes.

The next line to be altered is 2850 which should be changed to:

```
2850 DIM day$(8),sec(3,14)
```

To allow for the greater number of zones 3360 becomes:

```
3360 FOR Y=1 TO 3:FOR Z=1 TO 14
```

The DATA statements in lines 3420, 3430 and 3440 are extended to include the charges for the extra zones. If there are any changes in the times allowed per unit, it is here that these changes would be made. They now become:

```
3420 DATA 330,96,60,45,12,9,7.6,6.65,5.15,3.
55,3.8,2.65,2.15,0
3430 DATA 85,34.3,30,24,8,7.2,6.2,5.46,4.35,
2.9,3.05,2.25,2.15,0
3440 DATA 60,25.7,22.5,18,8,0,0,0,3,95,0,0,
0,2.15,0
```

Now we come to the extra procedures to support the rate finding and dialling errors. These should be numbered so that they are simply appended to the end of the original program:

```
DEF PROCdialerror
PROCborder1
 PRINTTAB(4,7)"Invalid dialling code,
please verify"
TIME=0:REPEAT
UNTIL TIME=300
dist=14:dist$="Error"
ENDPROC

DEF PROCrate1a
T%=hour
 IF T%<25 rate=1:rate$="CHEAP":ppr%=146:
ink%=129
 IF T%<20 rate=2:rate$="STANDARD":ppr%=1
47:ink%=132
 IF T%<8 rate=1:rate$="CHEAP":ppr%=146:i
nk%=129
unitsec=sec(rate,dist)
ENDPROC

DEF PROCrate1b
T%=hour*60+min
 IF T%<1441 rate=2:rate$="STANDARD":ppr%
=147:ink=132
 IF T%<1170 rate=1:rate$="CHEAP":ppr%=14
6:ink%=129
```

```
 IF  T%<870  rate=2:rate$="STANDARD":ppr%=
147:ink%=132
 IF  T%<420  rate=1:rate$="CHEAP":ppr%=146
:ink%=129
 unitsec=sec(rate,dist)
 ENDPROC

 DEF PROCrate1c
 rate=2:rate$="STANDARD":ppr%=147:ink%=1
32:unitsec=(rate,dist)
 ENDPROC

 DEF PROCalpha
 rate=1:rate$="Cheap":unitsec=(rate,dist)
 ENDPROC
```

When you have put all of this in, renumber the program again in increments of 10, and then the whole program will be neat and tidy, and ready to run. Don't forget to save your new program before running it.

NOTE: The complete and fully working program is included in its entirety on this month's magazine disc.

PROGRAM NOTES

I have followed the same general style of the original as far as possible. One extra safeguard has also been incorporated to overcome a problem that I experienced with the original program. When hitting Return to finish a call, the program did not always return to the same point for entering the dialling code but asked for the number instead. This was caused by PROCnumber looking for a Return immediately after entry into the input routine. A delay of .5 second was introduced at the beginning of PROCnumber which forestalled the double strike problem.

Some of the times allowed with overseas calls produced a further problem, as *STR$(unitsec)* would not return the value of certain decimal numbers, 6.2 printed out as 6.1999999999999, creating havoc with the screen layout. This was due to the fact that binary numbers do not readily convert to certain decimal fractions or vice versa. To overcome this the way in which *unitsec* was used in the display was changed with small alterations to three lines of PROCtime.

The first thing to consider with overseas calls was that the codes were often much longer than here in the U.K., as were the numbers, owing to the inclusion of area codes. For these reasons three lines of FNcheck1 had to be altered to accept longer codes and numbers. PROCrate had to be extended by fourteen lines, to sort out the different charge zones, as one international zone has a unified rate, others don't recognise weekends and there is another one that changes between cheap and standard charges at odd times. Another three versions of PROCrate1 had to be added to support the work of PROCrate; these are PROCrate1a, PROCrate1b and PROCrate1c. PROCrate1b is interesting in that the structure of the original PROCrate1 will not accept half hours - the Far East has change-overs at 2.30 and 7.30, therefore the day had to be reduced to minutes so that the half hours could be detected.

Within PROCdata, data had to be changed: Z had to be increased from 4 to 14, as did the array sec(3,4) to become sec(3,14). In the first line of PROCdef, 3 lines of data had to be extended to accommodate the seconds per unit of the international zones; some were dummy zeros to maintain the dimensions of the array, and there are also zeros as the last (14th) set of data which is used to support the freephone numbers. Also, *unitcharge* had to be made zero for the freephone and restored to 5.06 at any other time. This freephone patch caused one more problem: in PROCcost an exit had to be made in the first line because the Beeb does not like dividing by zero!

Another procedure was included to sort out invalid international codes as they would otherwise indicate a charge for "b" rate domestic calls. This is PROCdialerror which is called from the end of PROCdist. This puts a message on the screen for 3 seconds, and then returns the user to the input mode. Dublin numbers presented a different problem - being "0001" they were being detected as 'end data block' and created chaos initially with the mobile phone codes, at which rate they are charged. To overcome this, the 'end data block' was altered to "0000". The international dialling codes run into some 60 extra lines of code, and these came straight from the Phone Book. I found that programming two of the function keys to return "010" and "DATA" really speeded up typing in the data.

Sidways ROM Image Auto Loader

Following last month's article on loading ROM images, John Lasruk explains how to make your ROMs auto-loading.

I've read many magazine articles which aim to solve the problem of easily loading sideways ROM images into RAM. Among these were countless suites of Basic procedures, machine code routines and even a sideways ROM image which did the job, but only after it had itself been somehow loaded into paged RAM. The simplest solution of all must be to let the ROM image load itself.

Type in and save the following program. When run, this assembles some appropriate code below &3000, and then prompts for filenames, first for the "plain" incoming ROM image, then for the self-loading outgoing image. The image is loaded to &3000 and saved back from &2F3A, taking the extra code with it. Despite the fact that one extra sector per file is used by self-loading ROM images, 80 track DFS discs will still hold the same number of 16K images (12) as before, as will 40 track discs (6) and large format ADFS discs (39). To save further space, the program prompts for image length so that 8K ROMs won't use up unnecessary space. The default is 16K.

The routine works best with twin drives, so that incoming files and outgoing ones are on different discs. Just include the full path in the outgoing name (e.g.: :1.R.RomName). A single ADFS disc drive can be a problem, since switching discs isn't allowed. In this case, you should leave adequate room on the disc and (preferably) save the outgoing file to another directory. A large format ADFS disc should have room for 18 file transfers.

Finally, to use the amended ROM images, simply type:

```
        *RUN filename <RAM ID>
```

or:

```
        *filename <RAM ID>
```

where <RAM ID> is the decimal number of the RAM bank into which you wish to move the image. To be used as a star command, the file should be in the library directory, or $ directory if this hasn't been selected. The ID must be in decimal numerals only in the range 0-15. Digits are ignored after the second, and non-decimal input puts up an error message. Silly decimal input such as "97" will only load properly by sheer chance. (owing to the way in which the routine reads the parameters, "97" will be read as "1".)

The routine checks for RAM at <RAM ID> and gives another error message if it finds none. If everything works, you should get the message "BEEBUG: OK". The routine writes the appropriate ROM type byte to the table at &2A1, thus "turning it on", but since some ROMs require workspace, you may still have to press Ctrl-Break to initialise the software.

An autoloading ROM image will use main memory from &2F00 (approximately) to &7000, so it may overwrite other utilities or Basic programs. It is also unlikely to be compatible with an active co-processor or some shadow RAMs.

The code exits at several points using the "BRK" error handler. Error 255 indicates a successful transfer. Error 100 is a "Bad Parameters" error (which can also indicate no parameters), while error 101 indicates that the routine found no RAM at the designated location. These errors may be trapped by Basic and used within a ROM-loading menu program.

PROGRAM NOTES

The actual transfer of the ROM image to sideways RAM is a rather simple affair, consisting of paging in the appropriate RAM bank, and then moving up a copy of the image from &3000 to &8000. Once the slot is selected, and before any bytes are copied, I test for RAM by first writing, then reading back, the values 0 and 255.

A more difficult part is reading then making sense of the parameters. A visit with the Wizard of OSARGS tells me just where the parameters are located in memory (this varies, remember, with the length of the filename).

I translate the parameters from ASCII by subtracting 48 from them (ASC"0"=48). This gives me the opportunity to ignore any characters which are below the numeral "0" on the ASCII table. I store the result and test for values over 9 by subtracting 10 (if it isn't a negative value, it's not a numeral).

If the second byte is a Carriage Return, the parameter is one digit only and the appropriate value is passed to the main section of the program. If the second byte is not ASCII 13, I forget the first byte and simply add ten to the second byte's value and then pass that along. "66" (for example) is accepted as a parameter because the routine reads "66" as "16". But before any value is seen by the rest of the code, the top nibble is filtered out, so that 16 is finally seen as 0.

```
   10 REM Program AutoSid
   20 REM Version B1.21
   30 REM Author  John Lasruk
   40 REM BEEBUG  November 1990
   50 REM Program subject to copyright
   60 REM Error numbers indicate exit:
   70 :
  100 MODE 7
  110 FOR X=1 TO 2:VDU131,157,129,141
  120 PRINT SPC(4);"BEEBUG Sideways Self
Loader":NEXT
  130 FORY=2 TO 24:PRINTTAB(0,Y);:VDU132
,157,135:NEXT
  140 VDU28,3,22,36,5,21
  150 FOR X=1 TO 2:PROCassemble:NEXT
  160 VDU6:REPEAT:CLS
  170 PROCload:PROCsave
  180 UNTIL FNask<>121
  190 MODE7:END
  200 :
 1000 DEF PROCassemble
 1010 P%=&2F3A
 1020 [
 1030 OPT0
 1040 LDX #&70:LDY #0
 1050 LDA #1:JSR &FFDA
 1060 LDA(&70),Y:STA &76:INY
 1070 LDA(&70),Y:STA &77
 1080 LDA &76:JSR testvalue:STA &76
 1090 LDA &77:CMP #13:BEQ onedigit
 1100 JSRtestvalue:STA &77
 1110 CLC:LDA #10:ADC &77
 1120 STA &75:JMP start
 1130 .onedigit
 1140 LDA &76:STA &75
 1150 .start
 1160 LDX &F4:LDA &75:AND #15
 1170 STA &FE30:STA &F4:JSR testram
 1180 LDY #0:STY &70:STY &72
 1190 LDA #&80:STA &71:LDA #&30:STA &73
 1200 .transfer
 1210 LDA(&72),Y:STA(&70),Y
 1220 INY:BNE transfer
 1230 INC &71:INC &73:LDA &73
 1240 CMP #&70:BNE transfer
 1250 .return
 1260 TXA:PHA:LDX &75
 1270 LDA &3006:STA &2A1,X
 1280 PLA:STA &FE30:STA &F4
 1290 BRK:EQUB &FF:EQUS "BEEBUG:OK ":EQU
B 0
 1300 .testram
 1310 LDA #0:STA &8000:LDA &8000
 1320 CMP #0:BNEnoram
 1330 LDA #255:STA &8000:LDA &8000
 1340 CMP #255:BNE noram:RTS
 1350 .noram
 1360 STX &FE30:STX &F4
 1370 BRK:EQUB 101
 1380 EQUS "Not RAM":EQUB 0
 1390 .testvalue
 1400 SEC:SBC #48:BCC wrong
 1410 STA &79:SEC:SBC #10:BCS wrong
 1420 LDA &79:RTS
 1430 .wrong
 1440 BRK:EQUB 100
 1450 EQUS "Bad Parameters":EQUB 0
 1460 ]
 1470 ENDPROC
 1480 :
 1490 DEF FNask
 1500 INPUT''"Continue? "Q$
 1510 =(ASC LEFT$(Q$,1)) OR 32
 1520 :
 1530 DEF PROCload
 1540 INPUT"Name of ROM image to load?"'
in$
 1550 INPUT'"Size? (8/16, default 16) "
len
 1560 IF len=8:top$="5000" ELSE top$="70
00"
 1570 OSCLI "LOAD "+in$+" 3000"
 1580 ENDPROC
 1590 :
 1600 DEF PROCsave
 1610 INPUT'"Name of ROM image to save?"
'out$
 1620 OSCLI "SAVE "+out$+" 2F3A "+top$
 1630 ENDPROC
```

1st course

Better Programming

by Mike Williams

After some welcome diversions from other contributors I want to return to a subject which I consider to be vital, but which can prove so difficult to explain - the question of what constitutes 'good' programming. Despite the problems I have accumulated a number of examples which I hope will assist in this endeavour.

As readers will appreciate, we get sent many programs some of which end up in the pages of BEEBUG magazine. We therefore have the opportunity to see a wide variety of programming styles, some good, some bad. It is this question of what constitutes 'good' programming practice that I want to consider this month, by looking at some examples of how not to do it which have come my way recently. I hope to explain why these examples should be avoided, and to suggest better alternatives.

First I'd like to make a few general comments (and I am sure, repeat some of the things I have said before in articles in this series). One fundamental requirement of any well written program is that it should be *well structured*. This means that the program consists of a number (possibly quite a large number) of routines, each of which is well defined (i.e. when you read such a program it is easy to see the start and finish and hence content of each routine). A program will normally consist of a relatively short *main* program, which calls a number of these routines. Each routine may in turn call others. Thus the program contains a hierarchical structure in which each routine or function is clearly defined (see figure 1).

In BBC Basic, the best form to use for these routines is to define them as functions or procedures, preferably giving each a name which is reasonably descriptive of its function. An alternative, which is the traditional perquisite of Basic, is the use of the subroutine, using GOSUB <line number> to call it, and RETURN at the end of the subroutine definition to return to the point from which the subroutine was called.

The downside of subroutines is that jumping to a line number gives no clue as to the subroutine's function, and thus confusion and mistakes become more of a risk. Secondly, there is nothing to mark the specific start of a subroutine other than that its starting line number will be referenced in a GOSUB - not a very clear or satisfactory way of ensuring that a routine is clearly defined. In any case, procedures and functions also have the advantage of parameter passing, so let's agree that simple subroutines are undesirable, and concentrate on the use of procedures and functions from now on.

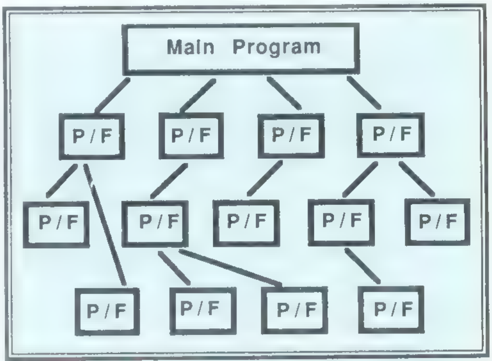


Figure 1. Typical hierarchical organisation of a well structured program using procedures and functions (P/F).

The other Basic instruction which is also considered less than desirable is the simple GOTO <line number>, and for the same reason. A GOTO can never tell you what function within the program is being jumped to. What is worse, is that as a program is developed, and problems are encountered, GOTO enthusiasts add more and more of these instructions to their program. Agreed, at the end of the day the program may work, but for anyone trying to trace a sensible path through the maze of GOTOs, the end result is nothing less than a nightmare. You will still find such examples in some of the programs which we publish in the magazine. I know it's not good practice, but on some occasions the effort which would be involved in changing the author's program is out of all proportion to its merits.

So our second principle of good programming is to avoid the use of GOTO if at all possible. In fact, however complex and lengthy a program, it is nearly always possible to program it without recourse to using GOTO, except in some instances of error trapping. The solution is (once again) to use procedures and functions.

The other aspect of programming which can cause problems is the use of loops. Virtually every program written will use a number of loops to repeat sequences of instructions as many times as needed. Sometimes the number of repetitions is known in advance, in which case a FOR-NEXT loop can be used; in others a REPEAT-UNTIL loop is a better choice, ensuring that the sequence of instructions will be repeated until some condition is satisfied. Again, either of these two loop constructions is better than a GOTO to form a loop.

Enough of the general principles; let's now consider three particular examples of what I consider to be bad programming practice. Here is a nested loop structure which I came across recently:

```
<n> REPEAT
    REPEAT
    ..................
    IF condition1 THEN GOTO n
    ..................
    UNTIL condition2
    UNTIL condition3
```

In the above, 'n' represents a line number, while *condition1*, *condition2* and *condition3* are all expressions which evaluate to either TRUE or FALSE. So, in outline we have two nested REPEAT-UNTIL loops. There is nothing wrong in that. The culprit is (of course) the GOTO, but not just because I have said the use of GOTO is undesirable.

When a program enters a loop, Basic makes a note of that fact. In the example above we enter first one loop and then another. If the GOTO is followed we jump *outside* of the loop, to line number 'n', where we effectively encounter two more REPEATs. Now as far as Basic is concerned, it is still inside the two nested loops already entered because no UNTIL has yet been met. It now encounters the start of two more loops (as it seems). If the IF statement is followed again, the same thing will be repeated, and Basic will think it has encountered yet two more loops, making six nested loops in all.

Obviously, if this process is repeated many times, Basic will think that we have nested a huge number of loops, and would ultimately run out of memory space to hold all the relevant information. Even if this were not to happen, Basic would ultimately continue with the rest of the program, still convinced that it was executing one or more loops. Further confusion can arise if another UNTIL is encountered, as this will then be matched with one of the original REPEATs. As I hope you can see, all manner of confusions can occur (but that's not to say that such a program will not appear to be working correctly - it's often when things go wrong that the trouble really starts).

So what's the solution? In principle, you should make sure that once Basic has entered a loop, it *always* makes a proper exit from that loop. The example above could be recoded as follows:

```
<n> REPEAT
    exit=FALSE
    REPEAT
    .................
    IF condition1 THEN exit=TRUE
    IF NOT exit THEN PROCrest
    UNTIL condition2 OR exit
    UNTIL condition3
```

This may not be the only solution, of course. What we have done is to introduce a variable (*exit*) which functions as a *flag*, i.e. it indicates whether or not we want to terminate the loop prematurely. Once *exit* has been set TRUE, we no longer wish to execute the instructions, whatever they were, which followed the original IF statement. So put these as a new procedure, which I have called PROCrest, and include a line such as that shown which ensures

Continued on page 34

Vouchers **worth £235** redeemable at the Computer Shopper Show

New horizons

December 6-9
Wembley Conference Centre
London

A CONFERENCE SPONSORED BY ACORN
(in association with the Computer Shopper Show)

Acorn is to hold a major four day end-user conference running alongside this year's Computer Shopper Show.

It has been designed to bring computer users bang up to date with technology and what the company's renowned research and development team has achieved of late. Attendees will also be provided with a fascinating glimpse into the future of computing.

The conference will allow end-users to –

- See all the latest Acorn innovations being exhaustively put through their paces.
- Hear from the experts who developed them, all acknowledged world experts in their fields.
- Gain an insight into the current projects on which Acorn is working to ensure it will remain in the vanguard of technology.

No matter where your particular interest in computing lies, the conference will provide you with a unique opportunity to learn more.

For the timetable offers something for everyone.
The conference itself is divided into two distinct seminar programmes - one running on the Thursday and Friday, the other on Saturday and Sunday.

Drop into a seminar of your choice – or attend them all – and you'll be given discount vouchers redeemable against purchases out on the exhibition floor.

There you will find up to 280 stands, including the massive Acorn Village, where the company will be flanked by all its leading third party suppliers.

Here then is YOUR chance to be not only entertained and informed but also to purchase computer Christmas presents at unbeatable prices. And it's all thanks to Acorn.

But don't leave it too late. Conference tickets are limited and will be allocated on a first come, first served basis.

Ensure your seat by completing and returning your application form TODAY.

Conference registration

Complete the registration form below. Sessions are priced at £7.50 each on Thursday and Friday and £5 each on Saturday and Sunday. Or buy a "Rover ticket" and attend all sessions - £30 per ticket for Thursday or Friday - £15 per ticket for Saturday or Sunday.

Entrance to the Computer Shopper Show at the door is £5 for adults and £3.50 for under 16's, but for Acorn conference delegates only we are offering a reduced rate at £3 per adult and £2 per child "Rover ticket" holders get in free!

Tick the boxes to indicate which sessions you will be attending:-
Please register me for the following sessions @ £7.50 each

Thursday/Friday 6 & 7 December

☐ CD Rom in Education
☐ Schema - A spreadsheet for the family
☐ Image processing
☐ PC emulating
☐ Music-Midi-Mania

Timetable of events

THURSDAY AND

FRIDAY

11.00 - 11.50 am
CD ROM IN EDUCATION
Speaker - *A spokesman for Next Technology.*

VOUCHER WORTH £50 for a Next Technology CD Rom Drive

12 NOON - 12.50 pm
SCHEMA - A SPREADSHEET FOR THE FAMILY
Speaker - *David Clare.*
He has worked in the computer industry for more than nine years and is best known as the head of one of Acorn's longest standing third party software houses, Clares

VOUCHER WORTH £30 for Schema

1.00 - 1.50 pm
IMAGE PROCESSING
Speaker - *Malcolm College.*

VOUCHER WORTH £40 for Wild Visions V9 Digitiser

2.00 - 2.50 pm
PC EMULATING AND OTHER OPERATING SYSTEMS
Speaker - *Ian Lynch.*
An expert on a variety of operating systems including IBM's PS2, MS dos, Unix and Risc OS, he is the curriculum development director for the City Technology Colleges Trust. In this role, he is responsible for supporting and developing curriculum innovation within colleges throughout the country

VOUCHER WORTH £30 for Acorn PC Emulator

3.00 - 3.50 pm
MUSIC-MIDI-MANIA
Speaker - *Mike Beecher.*
A graduate of the Royal College of Music, he is the managing director of ElectroMusic Research

VOUCHER WORTH £30 for EMR Studio 24

4.00 - 4.50 pm
RISC TECHNOLOGY IN THE 1990's
Speaker - *Mike Muller*
A key memberof the Acorn Risc development team.

VOUCHER WORTH £25 for Programmers Ref. Manuals

5.00 - 5.50 pm
DESKTOP PUBLISHING
Speakers - *Martin Chappell and James Lynn.*
Martin Chappell is the Art Editor of Car Magazine, the largest publication of its type in the UK

James Lynn is one of the chief designers and programmers on the Impression development team at Computer Concepts

VOUCHER WORTH £30 for Impression 2

SATURDAY AND SUNDAY

11.00 - 11.50 am
HYPERMEDIA FOR ALL
Speaker - *David Tee.*
The author of two of the original programs on the first welcome tape for the BBC Micro, he is the man behind the Genesis project

VOUCHER WORTH £20 for Genesis

12.00 noon - 12.50 pm
SCHEMA - A SPREADSHEET FOR THE FAMILY
(See details as for Thursday and Friday)

1.00 - 1.50 pm
PROGRAMMING MADE EASY WITH BASIC 5
(Including Dabs Compiler)
Speaker - *David Atherton.*
A co-founder of Dabs Press, he is well known as a regular contributor to leading computer titles

VOUCHER WORTH £30 for a Dabs Press ABC Compiler

2.00 - 2.50 pm
PC EMULATING AND OTHER OPERATING SYSTEMS
(See details for Thursday and Friday)

3.00 - 3.50 pm
MUSIC-MIDI-MANIA
(See details for Thursday and Friday)

4.00 - 4.50 pm
FASTER COMPUTING - *all the expansion options for the Archimedes*
Speaker - *Alex Van Someren.*
The technical director of Aleph 1, producers of the first commercially available ARM 3 upgrade, he will demonstrate the full potential of the Archimedes and discuss future possible enhancements

VOUCHER WORTH £50 for Arm 3 upgrade

5.00 - 5.50 pm
DESIGNING DOCUMENTS WITH THE HELP OF A COMPUTER
Speaker - *James Lynn.*
A leading light in desktop publishing technology, he is a chief designer and programmer at Computer Concepts.

VOUCHER WORTH £30 for Impression 2

Getting to the conference

Getting to Wembley Conference Centre is easy. All roads leading to the site are well signposted and the Centre is well served by buses and the tube (Jubilee & Metropolitan lines).

☐ Risc technology in the 90s
☐ Desktop Publishing

Please register me for the following sessions @ £5 each.

Saturday/Sunday 8 & 9 December

☐ Hypermedia for all
☐ Schema - A spreadsheet for the family
☐ Programming made easy with Basic 5
☐ PC emulating
☐ Music-Midi-Mania
☐ Faster computing
☐ Designing documents with the help of a computer

Please enrol me for ......sessions @ £7.50
Please enrol me for ......sessions @ £5.00
☐ Please send me a Rover ticket @ £30.00
☐ Thurs day ☐ Friday
☐ Please send me a Rover ticket @ £15.00
☐ Saturday ☐ Sunday
......Adult tickets @ £3 } only valid for Acorn
......Child tickets @ £3 } Conference delegates

I would like to pay by:
☐ cheque made payable to Blenheim Database
TOTAL.........
☐ Credit card Access Visa Expiry date
Card number:

Name

Address

Please return completed order form to
The Computer Shopper Show Ticket office,
Blenheim Database Exhibitions Ltd, PO Box 2,
Ellesmere Port, South Wirral L65 3EA.
Hotline 051 357 1736

that these instructions are called only if *exit* has not been set. One final word of warning here; if you do use a flag variable as here, then make sure that you set it initially (in this case FALSE) or the program will fail when it tests the conditions of the first UNTIL.

Here's another example of bad programming which I noticed quite recently, again involving REPEAT-UNTIL, though it could just as easily have applied to a FOR-NEXT loop:

```
DEF PROCbad
REPEAT
.......................
IF condition1 THEN UNTIL TRUE:EN
DPROC
.......................
UNTIL condition2
ENDPROC
```

In fact, the original example was even worse, as it used GOSUB and RETURN rather than a procedure. In some ways this is a similar situation to the previous one, with the need to make a premature exit from the loop (and hence the procedure) when *condition1* is TRUE.

In order to ensure that the loop is correctly terminated, the writer has ended up by including two UNTILs to go with the one REPEAT. In the case of a premature exit, the UNTIL is only there to exit from the loop formally, and no test is now required. Hence the UNTIL TRUE which *always* ensures a loop terminates (similarly, REPEAT-UNTIL FALSE ensures a loop *never* terminates).

However, it is undesirable that one REPEAT should have more than one UNTIL - each loop should have one entry point and one exit point, otherwise as program development proceeds it is very easy for mistakes to be introduced. The solution is again very similar to our first example, and that is to use a flag. So the procedure definition becomes:

```
DEF PROCbad
exit=FALSE
REPEAT
...........................
IF condition1 THEN exit=TRUE
IF NOT exit THEN PROCrest
UNTIL condition2 OR exit
ENDPROC
```

with PROCrest containing all the instructions previously following the IF statement. Note that to have written:

```
IF condition1 THEN ENDPROC
```

as an alternative would be equally bad, as it would mean an exit from the procedure while still within the REPEAT-UNTIL loop, with potentially even worse problems than in the first example.

The last example which I want to draw to your attention is quite different, though you might think of it as the reverse of the previous one, as it consists of two procedure 'starts' but only one ENDPROC:

```
DEF PROCa:LOCAL A%:A%=2
DEF PROCb:LOCAL A%:A%=3
........................
........................
ENDPROC
```

When calling PROCa, this definition relies on the fact that if Basic unexpectedly encounters a line beginning with DEF PROC, it just ignores that line. Thus if PROCa is called, A% is made local, and assigned a value of 2; if PROCb is called, A% is again made local but assigned a value of 3. As you might guess, it is not a good idea to mix up two procedures in this way.

In fact there is a very simple solution, by introducing a parameter into the definition. We thus get:

```
DEF PROCab(A%)
............................
............................
ENDPROC
```

The equivalent of PROCa would be called by writing:

```
PROCab(2)
```

and PROCb:

```
PROCab(3)
```

Furthermore, it is no longer necessary to declare any variable explicitly as local, as any variable used in the definition of a procedure's or function's parameters (such as A% above) is automatically treated as being local to the procedure.

Practical Assembler: Sorting with CALL

by Bernard Hill

Two months ago we started to look at the CALL statement in Basic, and how it can take added parameters for the assembler routine to pick up via a parameter block at &600. The aim over these three articles has been to write a complete utility which will sort an array (of strings, integers or reals) which follows the call. For instance:

```
      CALL sort,a(0),n%
```

where n% contains the number of elements of the array to sort, and a(0) would be the first element to take part in the sort. We discovered how the assembler code can detect whether the parameters are real, string or integer and how it finds out where they're stored. Last month we did some 'bottom-up' programming and developed a section of code which would find out whether the integers (or strings or reals) pointed to by zero page locations (*ptr1*) and (*ptr2*) are in ascending order or not, correctly handling all the different cases automatically.

This month we are going to complete the program, and if you typed in last month's part then you should delete all except lines 1410-1870 and keep these for merging (either with *SPOOL/*EXEC or a Basic utility such as Toolkit Plus) into this month's program to save you typing the complete listing.

A MACHINE-CODE SORT ROUTINE

Obviously we shall need an assembler sort routine. If (like many) you sometimes have difficulty getting even your Basic sort routines to work properly then take heart: we are going to do a bit of hand translation. Program 1 is a simple Bubble Sort straight out of the book (I know it's not particularly efficient but it's short) and we are going to translate this into assembler code, line by line, to give us our sort routine. Here it is:

```
10 INPUT N:DIM val(N):REM number of items
20 FOR i=0 TO N
30 val(i)=RND(1000):NEXT
40 PROCsort
50 FOR i=0 TO N:PRINT val(i)
60 NEXT:END
70 :
100 DEF PROCsort
110 IF N=1 THEN ENDPROC:REM no sort to do
120 FOR i=1 TO N
130 FOR j=N TO i STEP -1
140 IF val(j-1)>val(j) THEN x=val(j-1):
    val(j-1)=val(j):val(j)=x :REM swap
150 NEXT j
160 NEXT i
170 ENDPROC
```

Now assembler doesn't have FOR loops and compound THEN statements, so the first step (as mentioned in part 3 of this series) is to convert into 'Bad Old Basic With GOTOs'. We will also introduce a new variable k which is (j-1) because we will need it in assembler. Now PROCsort becomes:

```
100 DEF PROCsort
110 IF N=1 THEN ENDPROC
120 i=1
130 j=N
140 k=j-1:IF val(k)<val(j) THEN 160
150 x=val(k):val(k)=val(j):val(j)=x
160 j=k :   IF i<=j THEN 140
170 i=i+1 : IF i<=N THEN 130
180 ENDPROC
```

This of course should be test-run before we start converting it to assembler. One other point to mention before introducing the finished program is to note that we shall need quite a bit of two-byte integer arithmetic, so to make our code readable and short we define some macros for comparing, copying and adding two-byte numbers (lines 3000-5020 in the listed program).

Lines 130, 140 and 160 are targets for GOTOs in Basic, and we can keep this idea as we define labels for them. See lines 1220, 1230 and 1980 in the program below.

We need also to consider how we find where any element a(M) is in the array. The initialisation of the program will give us the location of the 'bottom' of the array, a(0) and since integers take 4 bytes for each element we will calculate an offset from the base which is always 4*M. So we also define a multiply-two-byte-integer-by-four routine in line 6000. Luckily strings also take four bytes per element (see last month's articles for the description of the 4-byte String Information Block) but reals take five, so we shall need to multiply by five in this case (see lines 1260-1370). Another subtle point is that the when one string needs to be exchanged with another then we need only swap their String Information Blocks as these contain the pointers to the actual strings. Thus when swapping during the sort we can treat strings and integers alike.

The variables used in the program are thus as follows:

ptr1 and *ptr2* are zero-page 2-byte locations which will normally point to two different elements of the array.

ptr3 and *ptr4* are two more zero-page locations which are used in the case of the string comparison (see last month's article).

ptype contains the parameter type byte.

base contains the address of the first element of the array.

N contains the length of the array.

I, *J* and *K* are the same as i, j, k of the Basic program bubble sort above.

K4 and *J4* are the offset from the base of the array to the element (=4*K and 4*J except for reals, when they are 5*K, 5*J).

In fact this program is far from optimal, and is not even an efficient implementation of a bubble sort, but it does illustrate well the process of bottom-up, top-down and Basic translation to write a moderately complicated piece of assembler program. It's also a lot faster than Basic at sorting!

```
   10 REM Universal sort
   20 REM Version B1.0
   30 REM Author  Bernard Hill
   40 REM Beebug  November 1990
   50 REM Program subject to copyright
   60 :
  100 PROCassemble
  110 INPUT"How many in the array";n%
  120 DIM x%(n%),x(n%),x$(n%),x 4*n%
  130 PRINT"1. Real"'"2. Integer"'"3. St
ring"
  140 REPEAT
  150 INPUT "Choose:"c%
  160 UNTIL c%>0 AND c%<4
  170 FOR i=0 TO n%
  180 IF c%=1 THEN x(i)=(RND(1000)-500)/
10
  190 IF c%=2 THEN x%(i)=RND(1000)-500
  200 IF c%=3 THEN x$(i)=CHR$(64+RND(26)
):FOR j=1 TO RND(20):x$(i)=x$(i)+CHR$(96
+RND(26)):NEXT
  210 NEXT:PRINT"Sorting...":TIME=0
  220 IF c%=1 THEN CALL sort,x(0),n%
  230 IF c%=2 THEN CALL sort,x%(0),n%
  240 IF c%=3 THEN CALL sort,x$(0),n%
  250 T=TIME:FOR i=0 TO n%
  260 IF c%=1 THEN PRINT x(i)
  270 IF c%=2 THEN PRINT x%(i)
  280 IF c%=3 THEN PRINT x$(i)
  290 NEXT
  300 PRINT"Sort time "T/100" sec"
  310 END
  320 :
 1000 DEF PROCassemble
 1010 DIM sort 1000
 1020 ptr1=&70:ptr2=&72
 1030 ptr3=&74:ptr4=&76
 1040 FOR opt=0 TO 2 STEP 2
 1050 P%=sort
 1060 [OPT opt
 1070 LDA &600:CMP #2:BEQ cont1:BRK:BRK
 1080 EQUS "Incorrect parameter count"
 1090 EQUB 0
 1100 .cont1
 1110 LDA &606:CMP #4:BEQ cont2:BRK:BRK
 1120 EQUS "Parameter #2 bad":EQUB 0
 1130 .cont2
 1140 EQUS FNcopy2(&601,base)
 1150 EQUS FNcopy2(&604,ptr1)
 1160 LDY #0:LDA (ptr1),Y:STA N
 1170 INY:LDA (ptr1),Y:STA N+1
 1180 CMP #0:BNE not0  \ sort if N<>0
 1190 LDA N:BNE not0:RTS   \ RTS if N=0
 1200 .not0 LDA &603:STA ptype
 1210 LDX #0:STX I+1:INX:STX I    \I=1
 1220 .line130 EQUS FNcopy2(N,J)
 1230 .line140 SEC                \K=J-1
 1240 LDA J:SBC #1:STA K
 1250 LDA J+1:SBC #0:STA K+1
 1260 \ now get the address of the Jth
 1270 \ element to ptr1 and the Kth
 1280 \ to ptr2 ready to compare
 1290 EQUS FNcopy2(J,J4):EQUS FNmul4(J4)
 1300 EQUS FNcopy2(K,K4):EQUS FNmul4(K4)
 1310 \ reals are 5 bytes each
 1320 LDA ptype:CMP #5:BNE over2
```

```
1330 EQUS FNadd2(J4,J,J4)      \J4=J4+J
1340 EQUS FNadd2(K4,K,K4)      \K4=K4+K
1350 .over2
1360 EQUS FNadd2(base,J4,ptr1)
1370 EQUS FNadd2(base,K4,ptr2)
1380 LDA ptype:CMP #4:BEQ ints
1390 CMP #129:BEQ strings
1400 :
1410 .reals
1420 LDY #1
1430 LDA (ptr1),Y:ROL A:BCS rneg1
1440 LDA (ptr2),Y:ROL A
1450 EQUS FNjcs(greateq)
1460 \ both positive
1470 DEY
1480 .loop LDA (ptr1),Y:CMP (ptr2),Y
1490 EQUS FNjcc(less)
1500 EQUS FNjne(greateq)
1510 INY:CPY #4:BMI loop
1520 EQUS FNjpl(greateq)
1530 .rneg1 LDA (ptr2),Y
1540 ROL A:BCC less
1550 \both negative
1560 DEY
1570 .loop LDA (ptr2),Y:CMP (ptr1),Y
1580 BCC less:BNE greateq
1590 INY:CPY #4:BMI loop:BPL greateq
1600 .ints LDY #3
1610 LDA (ptr1),Y:ROL A:BCS ineg1
1620 LDA (ptr2),Y:ROL A:BCS greateq
1630 \ both positive
1640 .loop LDA (ptr1),Y:CMP (ptr2),Y
1650 BCC less:BNE greateq
1660 DEY:BPL loop:BMI greateq
1670 .ineg1 LDA (ptr2),Y
1680 ROL A:BCC less
1690 \both negative
1700 .loop LDA (ptr2),Y:CMP (ptr1),Y
1710 BCC greateq:BNE less
1720 DEY:BPL loop:BMI less
1730 .strings
1740 LDY #3:LDA (ptr1),Y
1750 STA len \ length of string
1760 LDA (ptr2),Y:CMP len    \ length2
1770 BCS sover1:STA len    \ min length
1780 .sover1
1790 \ now get ptrs to strings
1800 LDY #0:LDA (ptr1),Y:STA ptr3
1810 INY:LDA (ptr1),Y:STA ptr3+1
1820 LDA (ptr2),Y:STA ptr4+1
1830 DEY:LDA (ptr2),Y:STA ptr4
1840 \ start scanning string
1850 .loop LDA (ptr3),Y:CMP (ptr4),Y
1860 BCC less:BNE greateq
1870 INY:CPY len:BNE loop:BEQ greateq
1880 .less CLC:JMP ov3
1890 .greateq SEC
```

```
1900 .ov3
1910 BCS line160
1920 \ now swap data of ptr1,ptr2
1930 LDY #3 \ for ints & strings
1940 LDA ptype:CMP #5:BNE swap:INY
1950 .swap LDA (ptr1),Y:PHA
1960 LDA (ptr2),Y:STA (ptr1),Y
1970 PLA:STA (ptr2),Y:DEY:BPL swap
1980 .line160
1990 EQUS FNcopy2(K,J)
2000 EQUS FNcomp2(I,J)
2010 EQUS FNjeq(line140)
2020 EQUS FNjcc(line140)
2030 INC I:BNE P%+5:INC I+1    \I=I+1
2040 EQUS FNcomp2(I,N)
2050 EQUS FNjeq(line130)
2060 EQUS FNjcc(line130)
2070 RTS
2080 :
2090 .ptype EQUB 0
2100 .base EQUD 0
2110 .len EQUB 0
2120 .N EQUW 0
2130 .I EQUW 0
2140 .J EQUW 0
2150 .K EQUW 0
2160 .J4 EQUW 0
2170 .K4 EQUW 0
2180 ]:NEXT
2190 ENDPROC
2200 :
3000 DEF FNcopy2(a,b):[OPT opt
3010 LDA a:STA b:LDA a+1:STA b+1:]:=""
3020 :
4000 DEF FNcomp2(a,b):[OPT opt
4010 LDA a+1:CMP b+1
4020 BNE P%+10:BCC P%+8
4030 LDA a:CMP b:]:=""
4040 :
5000 DEF FNadd2(a,b,c):[OPT opt
5010 CLC:LDA a:ADC b:STA c
5020 LDA a+1:ADC b+1:STA c+1:]:=""
5030 :
6000 DEF FNmul4(x):[OPT opt
6010 ASL x:ROL x+1:ASL x:ROL x+1
6020 ]:=""
6030 :
7000 DEF FNjcs(addr):[OPT opt:BCC P%+5
7010 JMP addr:]:=""
8000 DEF FNjcc(addr):[OPT opt:BCS P%+5
8010 JMP addr:]:=""
9000 DEF FNjeq(addr):[OPT opt:BNE P%+5
9010 JMP addr:]:=""
10000 DEF FNjne(addr):[OPT opt:BEQ P%+5
10010 JMP addr:]:=""
11000 DEF FNjpl(addr):[OPT opt:BMI P%+5
11010 JMP addr:]:=""
```

Some Comments on the New Master ROM

Derek Gibbons offers some personal comments on the value of the new Master OS ROM from Acorn.

It seems a pity that, even now, there has still been no *definitive* review of the new OS 3.5 ROM for the Master 128. This article is certainly not intended to be such, but it is hoped that the points raised here will be of interest to anyone contemplating purchase of the new ROM.

Reviews published so far vary from the inadequate to the factually inaccurate (but see BEEBUG Vol.8 No.7). Even Acorn, in the Master 128 System ROM User Guide, are less than helpful by stating: "The best cure for a (compatibility) problem is to use only facilities provided, documented and supported by Acorn". Unfortunately they omit to mention that they (Acorn) have moved the goal posts, and certain features documented in the Master Reference Manual now operate in a different manner. Some programs written in good faith, based on that original information, now fail!

One of the more disappointing aspects of the new OS ROM lies in its handling of the Master's extended character set. At first sight it appeared that it was going to be possible to enter these characters (Greek etc.) from the keyboard, but Ctrl-@ does *not* add 128 to the value of the next character typed (nowhere in the User Guide is it implied that it should), and it seems that there is still no easy and direct way to type in the extended characters.

The confusion appears to stem from the Master Compact where the 'CODE' key does indeed operate in this way (see BEEBUG Vol.8 No.10 p.46). This leads one to wonder if some reviewers ever *actually* try out the products under review. This is reminiscent of the early days of the BBC Micro, when reviewers insisted that CHR$139 in mode 7 was "START BOX", simply because that was part of the Ceefax Teletext specification, despite it never being implemented on the Beeb. [In those early days much was undocumented and just had to be guessed at, not always correctly of course.]

The reality of the situation is that ASCII values greater than 127, if placed in the keyboard buffer with *FX138, are treated as function key presses as before, but if preceded by an ASCII NUL, they are treated as part of the extended character set. This is *not* because 128 is added to the value (the value must *already* be greater than 127), but because the preceding NUL tells the operating system to treat the code as a character instead of as a function key.

It appears that extended characters can only be entered directly from the keyboard (i.e. other than by using *FX138,0,n or PRINT CHR$n) by using one of the following:

1. Redefining the base address of the numeric keypad with *FX238, but this gives only a limited range of characters which are not all consecutive.
2. Using *FX225,2 to cause the function keys to generate a NUL followed by the function key value, but again giving a restricted number of characters.
3. Putting control code sequences into function key definitions.
4. Redefining the Tab key with *FX219.
5. Using a machine code routine such as that described in BEEBUG Vol.8 No.10 p.46.

Of course, the main advantage of the new OS ROM in this respect is that, once the extended characters have been entered, they are displayed correctly by Basic, View, Edit, etc, and can even be printed appropriately if the printer's italic character set is re-defined (BEEBUG Vol.8 No.9 p.32).

The use of *FX225,2 highlights another problem area. Under the 'old' rules, this FX command caused function key f0 and above to generate actual ASCII values of 2 and upwards, and this was used by some programs as an easy way of detecting function key presses. Because the action of this command has now been changed from that originally documented, such programs no longer work. If the *FX225,2 can be located and changed to *FX225,3, such programs will then work after a fashion, but the function keys will be one position out of sync with their function key strip, and the action originally associated with f0 will no longer be available. For a complete cure it is therefore necessary to break into the program (which may very well be in machine code) and change the function key action table to match.

The use of control code sequences in function key definitions produces another problem. Under the old rules (again documented in the Master Reference Manual, but see also BEEBUG Vol.8 No.9 p.18), such function key definitions could be used in View, in conjunction with Ctrl and Shift (after a *FX228,1), to simulate pressing View's dedicated function keys and cursor keys, alone or in various combinations. With the new OS ROM, the situation has changed - Ctrl and Shift and any function key now produce the extended characters corresponding to the ASCII values generated by the function key definition instead of the equivalent key presses - i.e. a definition such as |!J|!K will generate CHR$138;CHR$139 instead of Delete followed by Tab. It does not appear to be possible to reverse this action, so these short-cut methods to synthesise multiple sequential key presses are now ruled out.

The redefinition of the Tab key can obviously produce only one new character at a time. Even so, it must be used with caution because a little-known bug appears to have been perpetuated all the way from OS 0.1, and is still present in the new OS ROM despite the claim that "all the known bugs in the standard MOS ROM have been fixed". This bug sometimes results in the character, produced by pressing the redefined Tab key, not corresponding to the ASCII value defined. ASCII values 0 - 64 give the correct effects; values 65 - 90 give characters 97 - 122; 91 - 96 are correct; 97 - 122 give 65 - 90; 123 - 127 are correct; 128 - 143 are treated as function key presses; 144 - 159 are correct; and 160 - 255 give 32 - 127! According tio Acorn, it is a feature of the way in which key codes are converted to chatacters.

One of the more serious aspects of the new OS ROM arises through the use of *FX151,78,127 followed by CALL!-4, to simulate a power-on reset. While this may not be fully supported by Acorn, it is widely used to initialise certain sideways RAM images. Unfortunately, if used with the new OS ROM, it completely CLEARS sideways RAM - thus defeating the object of the exercise! Giving Acorn the benefit of the doubt, it could be argued that on powering-up the micro, all sideways RAM would be clear; therefore, a power-on reset should have the same effect.

This also has implications with regard to *CONFIGURE. The Reference Manual states that *CONFIGURE commands are not actually implemented until Ctrl-Break is pressed. With the old OS ROM, the combination of *FX151,78,127 and CALL!-4 could also be used to simulate a Ctrl-Break and so implement any outstanding *CONFIGURE commands, but with the new OS ROM this combination again clears any sideways RAM. Fortunately, many *CONFIGURE actions can be implemented by just a simple Break and so CALL!-4 on its own will often suffice.

However, it appears that *CONFIGURE commands associated with filing systems (e.g. *CONFIGURE FILE 9), and possibly others, require something stronger than a simple Break. So far, *FX200,2 followed by CALL!-4 has worked in all situations tested, although on the face of it this should only additionally clear user memory!

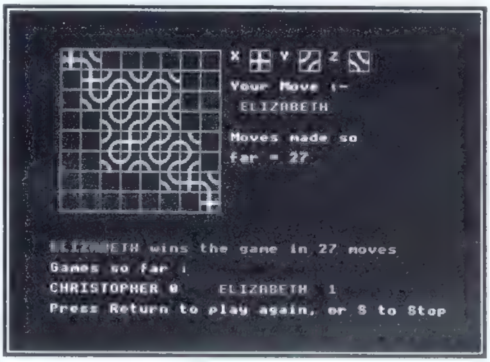
Finally, many programs exist which are not written according to Acorn's guidelines, and which directly access certain dedicated locations in. It is to be hoped that, in the future, when such programs are published, appropriate locations will be quoted not only for Basic II & IV, but also for the version of Basic (IVr32) in the new OS ROM (this was done with the EdiKit ROM, published earlier this year and now available on disc or EPROM, which was carefully rewritten and tested on this very point).

Any further information from readers, regarding the new Master ROM - problems encountered, fixes found - will be most welcome.

CURLY! An Educational Game

by David Williams

One of the problems that younger children have is to be able to visualise shapes and movements in their minds. As they get older so these skills increase, but different children often develop these at very different rates. In fact even some adults have a problem when using a map and going 'down' the page, and trying to decide if the next turn is a right or left one!



```
X   Y   Z
Your Move :-
 ELIZABETH

Moves made so
far = 27

ELIZABETH wins the game in 27 moves
Games so far :
CHRISTOPHER 0      ELIZABETH  1
Press Return to play again, or S to Stop
```

Diagram 1

The game of CURLY!, which is for two players, is designed to strengthen these skills of visualisation and imagination, as well as of strategy and anticipation. Although I have eight to twelve year olds in mind, and it fits in well with the qualities needed by the National Curriculum, it can be quite a demanding game for older players. It has the advantage that it is very easy to learn, even if it is harder to be successful! Even by yourself, you can draw some fascinating patterns.

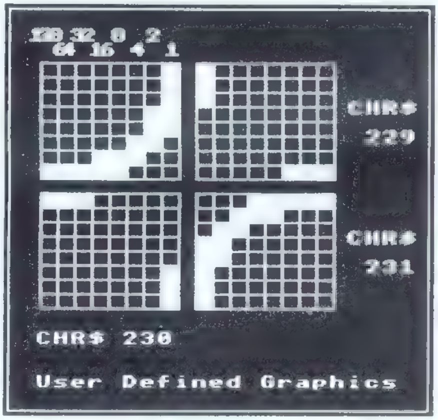


Diagram 2

To start, just type the program in and save it before you run it. The game starts at the top left-hand corner on an 8 x 8 grid, and finishes at the bottom right-hand corner. To get there you have a choice of three keys, each giving a different pattern which joins onto the previous one (see Diagram 1). With practice, you can drive your opponent off the board, or get to the winning square first.

PROGRAM NOTES

The program runs in mode 1, and the coding is fairly tight on a BBC B, leaving about 200 free bytes after several games. First we have to design the graphics, using characters 224 to 235, plus a 'special' CHR$253 for the opening title. Each of the X,Y, and Z patterns are made up of four defined characters, in two pairs to make a block of four stored, for example, as X$(1) and X$(2). See Diagram 2, and pages 170-172 in the User Guide.

Most of the coding is quite straightforward, but the tricky bit is after a player has selected his move, which then has to be checked (line 710 PROCcheck). The moves made are stored in the array *board(8,8)*, using the values 1, 2, or 3 for the three possible pattern choices, or zero if empty.

Meanwhile, the current move position is stored as X and Y, with the previous move as X1 and Y1. According to the choice, the check has to decide whether the next square is right, left, up or down, and hence works out the increment *addx*, *addy*, and also the sign (minus to the left, or up).

So the logic in line 1500 says, if the pattern is X, the *value* is 1, and if that is TRUE, *addx* = +1. If *value*=2 THEN *addy* = -1 ELSE if *value*=3 THEN *addy* = +1. On the whole I try not to use logic statements as for many people it makes it harder to follow the program, especially as TRUE = -1 in BBC Basic not +1 as one might hope!

So for instance, using P for Player, we need to have the value alternate between 1 and 2 for the two players. In some Basics, there is a SWAP function, but in BBC Basic we have to devise our own routine. This could be written as P = -2*(P=1) - (P=2) or P = (P*P + 1) MOD 4 but surely the statement IF P=1 THEN P=2 ELSE P=1 is much clearer, and uses the same number of bytes! However, if you really want to show the 'experts' a thing or two, how about P = 3 - P which is only 5 bytes! This is economic and simple, but again its intention is not immediately clear. Although I have used the IF P=1 ... statement at line 1260, and a similar one at line 1430 you cannot use it at line 1350. If you did, the ELSE would take effect when *lose* was FALSE; a basic trap (so to speak) that is easy for the unwary to fall into.

The remainder of PROCcheck is to test if you have gone off the board, or reached the end (lines 1410-1420). The REPEAT-UNTIL loop is needed as the new move may join together two previously unconnected bits of the pattern, so that the end of the path becomes several squares away. The check keeps repeating until it finishes at an empty square.

After the check is complete and the pattern is to be printed, I could not find a way to print using TAB without deleting part of the grid, so I have done a quick reprint, but only of the missing two lines, and not the whole grid (PROCreprint, line 1780). This is complicated by the fact that as the patterns are pairs of characters, the gap between the grid lines is also in steps of two.

The other feature that may be of interest is the introductory title routine which uses the powerful and much neglected POINT command. By printing the word CURLY! at a known position, the routine looks to see what the colour is at each pixel, and then prints a square block (the defined CHR$ 253) as part of the enlarged letter. Try changing the word CURLY! to HELLO! for example, and it will still work.

Another bit of logic! At line 2490, if the colour is cyan, N%=3 is TRUE and CHR$253 is printed. Otherwise CHR$254 is printed, but as it is not defined, you do not see very much!

If you do not have the monthly disc, I hope you find this program is worth the effort of typing it in, and especially if you can use it with children, you should find it of value, as well as fun.

```
   10 REM Program CURLY!
   20 REM Version B1.0
   30 REM Author  David Williams
   40 REM BEEBUG  November 1990
   50 REM Program subject to copyright
   60 :
  100 MODE1:ON ERROR GOTO 240
  110 PROCchr:PROCvars
  120 PROCtitle:PROCname
  130 PROCprep:PROCrules
  140 :
  150 REM Main Loop
  160 REPEAT:REPEAT
  170 PROCplay
  180 UNTIL win=TRUE OR lose=TRUE
  190 PROCend:PROCprep
  200 UNTIL stop
  210 CLS:PRINTTAB(12,12)"Game over. Bye
!":END
  220 REM End Main Loop
  230 :
  240 MODE7
  250 IF ERR<>17 REPORT:PRINTCHR$131" at
 line ";ERL
  260 END
  270 :
 1000 DEF PROCprep
 1010 IF stop THEN ENDPROC
 1020 win=FALSE:lose=FALSE:COLOUR(start)
:CLS:
 1030 temp$=player$(start)+" starts this
 time":len=LEN(temp$):tab=1+(40-len) DIV
 2
 1040 PRINTTAB(tab,20);temp$;TAB(10,22)"
Press Return to begin";
 1050 G=GET:CLS
 1060 player=start
 1070 moves=0:AX=1:AY=2:X=1:Y=3:X1=1:Y1=
1
 1080 FOR A=1 TO 8:FOR B=1 TO 8
 1090 board(A,B)=0
 1100 NEXT:NEXT
 1110 board(1,1)=1:COLOUR1
 1120 PRINTTAB(1,1);X$(1);TAB(1,2);X$(2)
;TAB(15,15);X$(1);TAB(15,16)X$(2)
 1130 PROCstart:PROCboard
 1140 ENDPROC
 1150 :
 1160 DEF PROCplay
 1170 COLOURplayer:PRINTTAB(19,6)SPC20;T
AB(19,6)player$(player)TAB(0,20);:PROCso
und(P%)
 1180 REPEAT Z$=GET$
 1190 UNTIL Z$>="X" AND Z$<="Z"
 1200 COLOUR1
 1210 IFZ$="X"THEN PRINTTAB(Y,X);X$(1);T
AB(Y,X+1);X$(2): board(AX,AY)=1
 1220 IFZ$="Y"THEN PRINTTAB(Y,X);Y$(1);T
AB(Y,X+1);Y$(2): board(AX,AY)=2
 1230 IFZ$="Z"THEN PRINTTAB(Y,X);Z$(1);T
AB(Y,X+1);Z$(2): board(AX,AY)=3
 1240 PROCreprint:PROCcheck
 1250 IF win=TRUE OR lose=TRUE ENDPROC
 1260 IF player=1 THEN player=2 ELSE pla
yer=1
 1270 PRINTTAB(24,11);moves
 1280 ENDPROC
 1290 :
 1300 DEF PROCend
 1310 VDU28,0,31,39,17,12:VDU26
 1320 IF win=FALSE AND lose=FALSE THEN E
NDPROC
 1330 COLOURplayer
 1340 PRINTTAB(0,20);player$(player);
 1350 IF lose=TRUE PRINT" LOSES the game
 by going"''"off the board!":player=3-pl
ayer
 1360 IF win=TRUE PRINT" wins the game i
n ";moves;" moves"
 1370 COLOUR1
 1380 score(player)=score(player)+1
 1390 FOR A=1 TO 4:PROCsound(P%):PROCsou
nd(Q%):NEXTA
 1400 PRINT'"Games so far :"''player$(1)
;" ";score(1);:COLOUR2:PRINT"    ";playe
r$(2);"  ";score(2)
 1410 COLOUR1:PRINT'"Press Return to pla
y again, or S to Stop":*FX15,1
```

```
 1420 G$=GET$:IF G$="S" THEN stop=TRUE
 1430 IF start=1 THEN start=2 ELSE start
=1
 1440 AX=0:AY=0:*FX15,0
 1450 ENDPROC
 1460 :
 1470 DEF PROCcheck:REM CHECK MOVE
 1480 value=board(AX,AY)
 1490 REPEAT
 1500 addx=-(value=1):addy=(value=2)-(va
lue=3)
 1510 IF X<X1 addx=-addx:addy=-addy
 1520 IF X=X1 PROCsameX
 1530 AX=AX+addx:X=X+2*addx:X1=X-2*addx
 1540 AY=AY+addy:Y=Y+2*addy:Y1=Y-2*addy
 1550 IF X<1 OR X>16 OR Y<1 OR Y>16 lose
=TRUE:ENDPROC
 1560 IF X=15 AND Y=15  win=TRUE:ENDPROC
 1570 value=board(AX,AY)
 1580 UNTIL value=0:REM an empty square
 1590 moves=moves+1
 1600 ENDPROC
 1610 :
 1620 DEF PROCsameX
 1630 temp=addx:addx=addy:addy=temp:REM
swap X & Y
 1640 IF Y<Y1 addx=-addx:addy=-addy
 1650 ENDPROC
 1660 :
 1670 DEF PROCsound(S%)
 1680 SOUND1,-10,S%,2
 1690 ENDPROC
 1700 :
 1710 DEF PROCboard
 1720 COLOUR1
 1730 K%=324:FOR A%=32 TO 548 STEP 64:MO
VE A%,K%+156:PLOT1,0,512:NEXT:REMVertLin
e
 1740 FOR A%=156 TO 672 STEP 64: MOVE32,
A%+K%:PLOT1, 512,0:NEXT:REM Horiz
 1750 PRINTTAB(18,4);"Your Move :-":PRIN
TTAB(18,9);"Moves made so";TAB(18,11);"f
ar = ";moves:
 1760 ENDPROC
 1770 :
 1780 DEF PROCreprint
 1790 A%=64+(Y-2)*32:MOVE A%,K%+156:PLOT
1,0,512:REMVertLine
 1800 A%=156+(7-X DIV 2)*64:MOVE32,A%+K%
:PLOT1, 512,0:REM Horiz
 1810 ENDPROC
 1820 :
 1830 DEF PROCstart
 1840 PRINTTAB(18,1);"X ";X$(1);" Y ";Y$
(1);" Z ";Z$(1):PRINTTAB(20,2);X$(2);SPC
3;Y$(2);SPC3;Z$(2)
 1850 K%=324:FOR A%=640 TO 960 STEP160:M
OVEA%,K%+604:PLOT1,64,0:PLOT1,0,64:PLOT1
,-64,0:PLOT1,0,-64:NEXTA%
 1860 ENDPROC
 1870 :
 1880 DEF PROCname
 1890 FOR A%=1 TO 2
 1900 COLOUR A%:PROCsound(P%):PRINTTAB(0
,20)SPC40:PRINTTAB(0,20)"Player ";A%;"'s
 name please: ";:INPUT" "player$(A%)
 1910 player$(A%)=LEFT$(player$(A%),11)
 1920 NEXTA%
 1930 PROCsound(P%):COLOUR1
 1940 PRINTTAB(0,20)"Do you want to see
the rules? (Y/N) ":G$=GET$
 1950 ENDPROC
 1960 :
 1970 DEF PROCrules
 1980 IF G$<>"Y" THEN ENDPROC
 1990 PRINTTAB(3,19)"The aim of CURLY! i
s for ";player$(1)
 2000 PRINT"to make a  continuous  curly
  line from"
 2010 PRINT"the top corner to the bottom
 corner and"
 2020 PRINT"use only the X,Y, and Z keys
."
 2030 PRINT'TAB(3)player$(2);" has to tr
y and stop "'TAB(3)player$(1);", or get
there first!"
 2040 PRINT'"   If either player goes  o
ff the board"
 2050 PRINT"then he or she has lost."
 2060 PRINT'"   Press Return to start."
 2070 G=GET
 2080 ENDPROC
 2090 :
 2100 DEF PROCvars:REM Define Variables
 2110 VDU23;8202;0;0;0;:REM Cursor OFF
 2120 REM Define logical colours
 2130 VDU19,1,3,0,0,0
 2140 VDU19,2,6,0,0,0
 2150 COLOUR1:GCOL1,2
 2160 *FX202,32
 2170 REM Caps lock on
 2180 DIMX$(2),Y$(2),Z$(2),board(8,8),sc
ore(2),player$(2)
 2190 X$(1)=CHR$224+CHR$225:X$(2)=CHR$22
6+CHR$227
 2200 Y$(1)=CHR$228+CHR$229:Y$(2)=CHR$23
0+CHR$231
 2210 Z$(1)=CHR$232+CHR$233:Z$(2)=CHR$23
4+CHR$235
 2220 score(1)=0:score(2)=0:start=1:win=
FALSE:lose=FALSE:stop=FALSE
 2230 P%=177:Q%=165:R%=148:REMsound pitc
h
 2240 ENDPROC
 2250 :
 2260 DEF PROCchr:REM Define chr$
 2270 VDU23,224,1,1,1,1,1,1,1,127
 2280 VDU23,225,128,128,128,128,128,128,
128,255
 2290 VDU23,226,127,1,1,1,1,1,1,1
 2300 VDU23,227,255,128,128,128,128,128,
128,0
 2310 VDU23,228,1,1,1,3,3,6,28,248
 2320 VDU23,229,128,128,128,0,0,0,0,7
 2330 VDU23,230,224,0,0,0,0,1,1,1
 2340 VDU23,231,31,56,96,192,192,128,128
,128
 2350 VDU23,232,1,1,1,0,0,0,0,224
 2360 VDU23,233,128,128,128,192,192,96,5
6,31
 2370 VDU23,234,248,28,6,3,3,1,1,1
 2380 VDU23,235,7,0,0,0,0,128,128,128
 2390 VDU23,253,254,254,254,254,254,254,
254,254
 2400 ENDPROC
 2410 :
 2420 DEF PROCtitle:REM BIG LETTERS
 2430 hrz%=1024:vrt%=636:W%=-1:GCOL0,3
 2440 MOVE0,0:PLOT0,hrz%,vrt%:VDU5:PRINT
"CURLY!":VDU4:COLOUR2
 2450 FOR V%=1 TO 6
 2460 FOR Y%=0 TO 7:FOR X%=0 TO 6+(V%=6)
 2470 REM (V%=6) is a bodge to stop last
 column printing!
 2480 N%=POINT(hrz%+X%*4,vrt%-Y%*4)
 2490 PRINTTAB(W%+X%,4+Y%)CHR$(254+(N%=3
))
 2500 NEXT:NEXT
 2510 hrz%=hrz%+32:W%=W%+7
 2520 NEXT
 2530 PRINTTAB(2,12)"CURLY!"
 2540 GCOL1,2
 2550 ENDPROC
```

Producing 8mm Video Tape Inlays

Paul Timson describes the necessary modifications to convert his programs for producing audio tape inlays into the format for 8mm video tape.

In January/February 1989 BEEBUG published a set of programs which could be used for creating, editing and printing inserts suitable for audio cassette inlays. In response to requests from readers, the coding given below converts the original programs so that the results are suitable for 8mm video tape cassettes.

The original programs were published in BEEBUG Vol.7 No.8, and you will need copies of these programs. Alternatively, completely updated programs are included on this month's magazine disc. The disc, as before, also contains a loader program which creates a help file which can be invoked while running the editor. However, this is not essential.

There are two programs to update, the Editor and the Printer. In each case the relevant lines listed here need to be added to the original program. This is best done by saving the new code in a spool file, loading in the original program, and then using *EXEC to append the spooled amendments (refer to the User Guide for more details). Listing 1 should be added to the Editor program, and the coding of Listing 2 to the Printer program. You will also need the Menu program from the original - this is unchanged.

USING THE PROGRAMS

Always start by running the Menu program. This displays the following three options:

1. Edit new or existing tape inlay
2. Print out tape inlay
3. Exit tape inlay producer

On selection of the editor an extensive help list is available which provides full explanation of the editing commands (supplied on disc only).

EDITOR

The editor program allows tape inlays to be loaded, entered, edited and subsequently saved for printing or further editing. The editor initially displays a tape inlay, which is twenty two lines deep and sixty one characters wide. This is where the programmes that appear on the video tape are entered. Below this is a box labelled "Title", this being where the video title is entered.

At any time (if the help file is available) help on all the commands can be called up by pressing the 'H' key. Under direction of the help screen, cells can be inserted, deleted, copied, moved and edited. See below for the complete list. When saving and loading video tape inlays a disc catalogue is displayed to aid filename selection.

PRINTER

The printer program prints out video tape inlays which have been produced by the editor, and a disc catalogue is displayed to aid filename selection. The printer codes used are for Epson compatible printers, but can easily be modified to suit other printers by altering the variables in PROCinit.

The variables are as follows:

dc$	- Double-strike, condensed mode
de$	- Double-strike, enlarged mode
oe$	- One-eighth inch line-space
ul$	- Underline mode on
xul$	- Underline mode off
init$	- Initialise printer
bell$	- Sound printer bell

EDITOR COMMAND LIST

E - Edit line occupied by cursor
T - Edit Video cassette title

Cursor keys to move - Copy and Delete to delete forwards and backwards. When editing lines (i.e. not titles) a dash (hyphen) typed before the item means it will be centred and underlined, whereas a plus sign will make it underlined, but still left justified.

Cursor keys - Move cursor
Delete key - Delete left
Copy key - Delete right

L - Load previous tape inlay
S - Save current tape inlay
I - Insert a blank cell at the cursor
D - Delete cell at cursor
C - Copy cell to another position
M - Move cell to another position

When copying or moving a cell - press the 'C' or 'M' key on the cell to be copied/moved, and then move the cursor to the destination and press Return to continue or the Spacebar to exit.

***** - Catalogue disc
Z - Zap tape inlay
X - Exit program

Listing 1

```
   10 REM Program 8MM Tape Inlay Editor
  140 PRINTTAB(48,25);"Function (H is He
lp) :-";SPC(10);TAB(72,25);
  170 PRINTTAB(48,25);SPC(23);TAB(48,25)
;"Are You Sure? ";
 1010 DIMtr$(23)
 1120 PRINTTAB(17,0);"8MM Tape Inlay Pro
ducer - By Paul Timson"
 1130 PRINTTAB(5);"+";STRING$(62,"-");"+
"
 1140 FORad%=1TO22:PROCline(ad%):NEXTad%
 1150 PRINTTAB(5);"+";STRING$(40,"-");"+
";STRING$(21,"-");"+"
 1160 PRINTTAB(5);"|Title="+t1$+STRING$(
34-LEN(t1$)," ")+"|"
 1170     REM
 1220     REM
 1230 PRINTTAB(4,ad%+1);" |";tr$(ad%);ST
RING$(62-LEN(tr$(ad%))," ");"|"
 1240 IF pos%=ad% PRINTTAB(4,ad%+1);"*"
 1280 PRINTTAB(5,30);"Enter programme ti
tle :- ";
 1290 y%=pos%+1
 1300 x%=6
 1320 PROCalt(x%,y%,62,1)
 1365 IFtem$="C" OR tem$="c" flag=1 ELSE
 flag=0
 1420 te$=tr$(pos%)
 1422 from$=tr$(from%)
 1425 IF flag=1 AND tr$(pos%)<>"" PROCin
s
 1430 tr$(pos%)=from$
 1470 IF tr$(22)<>"" GOTO1490
 1480 FORad%=22TOpos%+1STEP-1:tr$(ad%)=t
r$(ad%-1):NEXTad%:tr$(pos%)=""
 1520 FORad%=pos%+1TO22:tr$(ad%-1)=tr$(a
d%):NEXTad%
 1530 tr$(22)=""
 1575 IFtem$=CHR$(32)GOTO1630
 1590 to%=pos%
 1600 pos%=from%:PROCdel:IF to%>from% po
s%=to%:PROCins:tr$(to%)=te$
 1610 IF to%<from% pos%=to%+1:PROCins:tr
$(to%+1)=te$
 1620 pos%=to%+(to%>from%)
 1670 tem%=ASC(tem$):IF tem%>=138 AND te
m%<=139 GOTO 1720
 1700 tem%=INSTR("HhLlSsIiDdCcEeXxZz**Mm
Tt",tem$):IF tem%=0 GOTO 1660
 1730 IFtem%=139PRINT"Up"ELSEIFtem%=138P
RINT"Down"
 1740 IFtem%<138GOTO1780
 1760 IFtem%=139ANDpos%<>1ANDpos%<>24pos
%=pos%-1ELSEIFtem%=138ANDpos%<>22ANDpos%
<>46pos%=pos%+1
 1790 IFtem$="D"PRINT"Delete":PROCdel EL
SE IFtem$="C"PRINT"Copy":PROCcopy ELSE I
Ftem$="M"PRINT"Move":PROCmove ELSE IFtem
$="T"PRINT"Title":PROCed
 1900 ad%=LEN(var$):IF at%=max%+1OR(at%=
max%ANDtyp%=1AND(LEFT$(var$,1)<>"-"ANDLE
FT$(var$,1)<>"+")=-1)at%=at%-1
 1910 PRINTTAB(x%,y%);var$;SPC(max%-LEN(
var$)):PRINTTAB(x%+at%-1,y%);
 2000 IFad%=max%OR(ad%=max%-1ANDtyp%=1AN
D(LEFT$(var$,1)<>"-"ANDLEFT$(var$,1)<>"+
")=-1AND(at%<>1OR(tem$="-" OR tem$="+")=
0)=-1)=-1 GOTO 1920
 2070     REM
 2080     REM
 2090     REM
 2100     REM
 2110 DEFPROCed
 2120 PRINTTAB(5,30);"Enter cassette tit
le:- ";
 2130 var$=t1$
 2140 PROCalt(12,25,34,2)
 2150 t1$=var$
 2185 *DIR V
 2270 FORad%=1TO22
 2300 IFtyp%=1PRINT#X%,t1$,pos%ELSEINPUT
#X%,t1$,pos%
 2325 *DIR $
 2475 *DIR V
 2535 *DIR $
```

Listing 2

```
   10 REM Program 8MM Tape Inlay Print
  150 PRINTTAB(30,0);"PRINT 8MM TAPE INL
AY"'TAB(30,1);STRING$(20,"-")'
  155 *DIR V
  165 *DIR $
  170 PRINTTAB(2,26);"Enter 8MM Tape Inl
ay To Print - <Escape> Exits"
 1010 DIMlist$(23)
 1200 PRINTinit$;dc$;oe$;"+";STRING$(63,
"-");"+"
 1210 PRINT de$;" ";SPC(INT((37-LEN(t1$)
)/2));ul$;t1$;xul$;SPC(37-LEN(t1$)-INT((
37-LEN(t1$))/2));" "
 1220      REM
 1230 st%=1:fin%=17:PROCindex
 1235 PRINTxul$;"|";STRING$(63,"-");"|"
 1240 PRINT"|";STRING$(63," ");"|"
 1250      REM
 1260 PRINT de$;" ";SPC(INT((37-LEN(t1$)
)/2));ul$;t1$;xul$;SPC(37-LEN(t1$)-INT((
37-LEN(t1$))/2));" "
 1270 PRINTdc$;"|";STRING$(63," ");"|"
 1280 PRINTul$;"|";STRING$(63," ");"|"
 1290      REM
 1300      REM
 1310      REM
 1320      REM
 1330 st%=18:fin%=22:PROCindex
 1400 IF st%=1 PRINT"|";STRING$(63," ");
"|"
 1420 IFad%=22PRINTul$;:txul$=xul$:xul$=
""
 1430      REM
 1440 IFLEFT$(list$(ad%),1)="-"temp$=RIG
HT$(list$(ad%),LEN(list$(ad%))-1):temp$=
STRING$(INT((63-LEN(temp$))/2)," ")+ul$+
temp$+xul$+STRING$(63-LEN(temp$)-INT((63
-LEN(temp$))/2)," ")
 1445 IFLEFT$(list$(ad%),1)="+"temp$=" "
+ul$+RIGHT$(list$(ad%),LEN(list$(ad%))-1
)+xul$+STRING$(63-LEN(list$(ad%))," ")
 1450 IFLEFT$(list$(ad%),1)<>"-" AND LEF
T$(list$(ad%),1)<>"+"temp$=" "+list$(ad%
)+STRING$(63-LEN(list$(ad%))-1," ")
 1460      REM
 1470      REM
 1480 PRINT"|";temp$;"|"
 1495 IF ad%=22 xul$=txul$
 1525 *DIR V
 1540 FORad%=1TO22
 1570 INPUT#X%,t1$,pos%
 1585 *DIR $
```

Points Arising....Points Arising....Points Arising....Points Arising....

FIRST COURSE: UNDERSTANDING DATA FILES (Vol.9 No.5)

There were some inaccuracies in this particular article. Just over halfway down page 45 it was stated that the exponent was &77-&80 (correctly) giving an answer of -3 (which is incorrect). It should be -9 (decimal) resulting in the subsequent moving of the decimal point 9 places right (not 3).

The concluding sentence of the same section gives the final result as 1/8+1/16=3/16=0.1875. This should have read (according to the binary fraction above) as 1/16+1/32=3/32=0.09375. We are sorry for any confusion that may have resulted from this, and our thanks to the sharp eyes of Mr.Fay of Doncaster who spotted these inaccuracies.

MONIX: A MACHINE CODE MONITOR (Vol.9 No.4)

There was a mistake in the printing of the listing for part 1 of this program. The last instruction in line 1510 should read:

```
BNE lp1
```

(and not BElp1 as printed). Our apologies for this error.

You may also notice that the label *retadr* is defined twice, once at line 1090 and again at line 1120. This is unnecessary - only one such definition is need, and either may be omitted, though no damage is caused by leaving the program as it is in this respect.

ADFS Disc Sector Editor (Part 2)

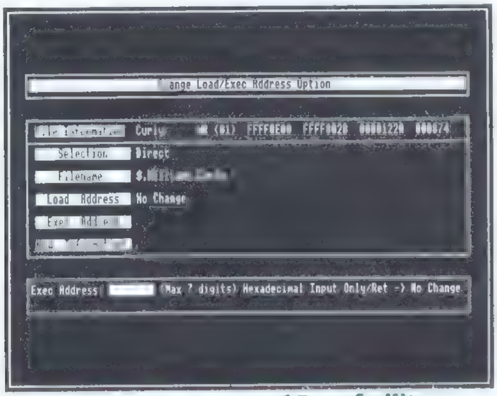
This month we complete Stefano Spina's Disc Sector Editor with a comprehensive disc recovery feature plus other options.

The first step is to append part two of this program (listed here) to the listing given last month. To do this type in Listing 1 and save it to disc. Now save a spooled version using the *SPOOL command. Next, load the program from part one and append the spooled version of part two to it using the *EXEC command. Now save the completed program, using a new name to avoid overwriting the existing files (in case you have made any mistakes). We will refer to the whole program as *DiscScan*.

Part two adds the procedures for the remaining three main menu options:

1. Recovery
2. Load/Exec Address Change
3. Command Page

We will deal with the last two first, as these are quite simple, and then devote the remaining space to discussing the use of the disc recovery option.



```
ange Load/Exec Address Option

                 Curly     (01)  FFFF0E00  FFFF0020  00001220  000874
Selection        Direct
Filename         $.
Load  Address    No Change
Exec  Address

Exec Address              (Max 7 digits) Hexadecimal Input Only/Ret -> No Change
```

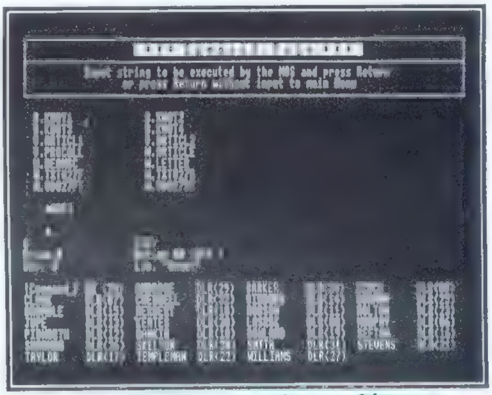
Using the Command Page facility

LOAD/EXEC ADDRESSES

When a file is saved to disc, it is given a *load* address (which determines the default load address for the file), and an *execution* address (which determines the point at which execution will start if the file is loaded as a program - *RUN <name>). It is sometimes desirable (even essential) to change these two addresses for an existing saved file, and that is what this option of DiscScan does. Either or both addresses can be changed. Simply enter the new address when prompted (in hex preceded by '&') in each case - just pressing Return keeps the existing address.

COMMAND PAGE

This simply provides an opportunity for the user to enter any star command. Pressing Return in response to the "*" prompt returns the user to the main menu.



```
     string to be executed by the MOS and press Return
        or press Return without input to main Menu

TAYLOR   DLR(17)   TEMPLEMAN   DLR(22)   WILLIAMS   DLR(27)   STEVENS
```

Changing Load/Exec addresses

DISC RECOVERY

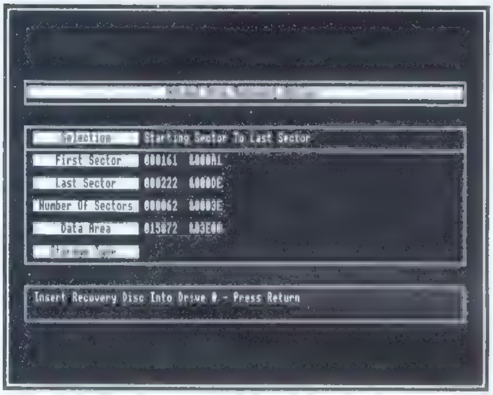
The purpose of this option is to permit, as far as possible, the recovery of a file from a disc which has become corrupted. Corruption can directly affect one or more sectors of a file, or the file directory on disc, preventing normal file access. The recovery option allows one or more sectors to be read from disc and then resaved. The Disc Recovery option uses sideways RAM to hold the disc sectors, and as a result there is a choice of either *(N)ormal* recovery (where the number of sectors does not exceed 255, the maximum which can be held in sideways RAM), and *(E)xtended* recovery for more than 255 sectors (up to a maximum of 2590).

The data area (that is the set of disc sectors to be recovered) can be specified in one of three ways:

1. By specifying start and end sector numbers
2. By specifying a start sector and the number of sectors
3. By specifying a start sector and the number of bytes.

Inputs can be either in decimal or in hexadecimal (preceded by '&'). To determine any of these parameters, first use the editor (described last month) to identify the relevant sectors. If the Normal recovery option has been chosen, then the sectors once read from the disc can be saved as a named file on another (or the same) disc, or the sectors can be saved offset by a specified number of sectors from the original starting point, again to the same or a different disc. In the case of Extended recovery, the sectors read can only be saved to a new named file.

When disc recovery takes place to a named file, DiscScan also provides an opportunity to change the load and/or the exec addresses of the new file. Once a file has been recovered, the editor can be used again to edit any individual bytes which may still be incorrect, and to complete the recovery procedure.



```
[illegible]
Selection          Starting Sector To Last Sector
First Sector       000161  &000A1
Last Sector        000222  &000DE
Number Of Sectors  000062  &0003E
Data Area          015872  &03E00
[illegible]

Insert Recovery Disc Into Drive 0 - Press Return
```

Entering parameters for disc recovery

```
  10 REM >PartTwo
 5000 :
 5010 DEFPROCaddress(M%)
 5020 LOCALL%,S%:L%=0:S%=0
 5030 CLS:PROCoption(7):PROCstbx(0,14,79
,26,0):PROCstbx(0,16,79,0,1)
 5040 PROCcolor(1):PRINTTAB(2,15)"File I
nformation"
 5050 PRINTTAB(2,17)"    Selection    ":
PRINTTAB(2,19)"    Filename     "
 5060 PRINTTAB(2,21)"  Load  Address  ":
PRINTTAB(2,23)"  Exec  Address  "
 5070 PRINTTAB(2,25)"New  Information ":
PROCcolor(0)
 5080 IF M%=0 ELSEPRINTTAB(20,17)"From R
ecovery Option":GOTO5130
 5090 PRINTTAB(20,17)"Direct"
 5100 PROCinput1(7):IF LEFT$(sr$,1)<>"$"
 path$="$."+sr$ ELSEpath$=sr$
 5110 PROCinput1(8):file$=path$+"."+sr$
 5120 IF NOTFNfound(file$) PROCalm(9):GO
TO5100
 5130 PRINTTAB(20,19)file$:VDU31,20,15:O
SCLI("INFO "+file$)
 5140 PROCinput1(9):IF sr$=CHR$13 PRINTT
AB(20,21)"No Change" ELSE IF LEFT$(sr$,1
)<>"&" PROCalm(10):GOTO5140 ELSEL%=sr%:P
RINTTAB(20,21)sr$
 5150 PROCinput1(10):IFsr$=CHR$13 PRINTT
AB(20,23)"No Change" ELSE IF LEFT$(sr$,1
)<>"&" PROCalm(10):GOTO5150 ELSES%=sr%:P
RINTTAB(20,23)sr$
 5160 IFL%<>0 PROCloadexec(file$,L%,2)
 5170 IFS%<>0 PROCloadexec(file$,S%,3)
 5180 VDU31,20,25:OSCLI("INFO "+file$):P
ROCget1(CHR$13,4)
 5190 IF M%=1 rec%=TRUE
 5200 ENDPROC
 5210 :
 5220 DEFPROCcomm:VDU22,128:LOCALA$
 5230 PROCstbx(0,0,79,5,0)
 5240 PROCstbx(0,2,79,0,1)
 5250 PROCcolor(1):PRINTTAB(20,1)"  * *
*  C o m m a n d   P a g e  * * *  "
 5260 PROCcolor(0):PRINTTAB(11,3)"Input
string to be executed by the MOS and pre
ss Return":PRINTTAB(18,4)"or press Retur
n without input to main Menu":VDU28,0,30
,79,6:CLS
 5270 REPEAT
 5280 INPUTLINETAB(4)"*"A$
 5290 IF LEFT$(A$,3)="DIR" PRINT:OSCLIA$
:OSCLI"." ELSE IF A$="0"ORA$="1" OSCLI"M
OUNT"+A$:OSCLI"." ELSE IF A$="^" OSCLI"D
IR^":OSCLI"." ELSE IF A$="$" OSCLI"DIR $
":OSCLI"." ELSE OSCLIA$
 5300 PRINT
 5310 UNTILA$="":VDU26
 5320 ENDPROC
 5330 :
 5340 DEFFNdirck(F$)
 5350 LOCALA$,B$,A%,B%,C%,D%:A%=LENF$
 5360 IF ((LEFT$(F$,1)=".")OR(RIGHT$(F$,
1)=".")OR(LEFT$(F$,1)="$"ANDA%>1ANDLEFT$
(F$,2)<>"$.")) PROCalm(11):=FALSE
 5370 B%=1:C%=0:D%=TRUE
 5380 REPEAT:A$=MID$(F$,B%,1):IF A$="."
C%=B%
 5390 IF (A$<>"."AND(B%-C%)>10)OR(A$="$"
ANDB%>1)OR(A$="."ANDMID$(F$,B%+1,1)=".")
 D%=FALSE:PROCalm(11) ELSEB%=B%+1
 5400 UNTILB%>A%ORNOTD%
 5410 =D%
 5420 :
 5430 DEFFNfound(F$)
 5440 LOCALA%,B%,C%,X%,Y%,Z%:DIMB%50,C%2
 5450 $B%=F$:C%?0=B%:C%?1=B%DIV256:X%=C%
 5460 Y%=C%DIV256:A%=&05:Z%=USR(&FFDD)AN
D&FF
 5470 IFZ%>0=TRUE ELSE=FALSE
 5480 :
 5490 DEFPROCget1(G$,M%)
 5500 LOCALA$,S$:PROCin1:VDU7
 5510 IF M%=1 S$="(S)tart Sector  (N)umb
er Of Sectors   (D)ata Area   (E)xit"
 5520 IF M%=2 S$="Insert Recovery Disc I
nto Drive 0 - Press Return"
 5530 IF M%=3 S$="(S)ector By Sector Wri
ting   (F)ile Saving"
 5540 IF M%=4 S$="Changes Executed - Pre
ss Return"
 5550 IF M%=5 S$="Change Load/Exec Addre
```

```
ss (Y/N)"
 5560 IF M%=6 S$="E(x)tended Area Recove
ry   (N)ormal Area Recovery   (E)xit"
 5570 IF M%=7 S$="Insert a New Formatted
 Disc Into Drive 0 - Press Return"
 5580 PRINTTAB(2,29)S$:PROCcurs(0)
 5590 REPEAT:A$=GET$:UNTILINSTR(G$,A$)<>
0
 5600 PROCin1:get%=ASCA$
 5610 ENDPROC
 5620 :
 5630 DEFPROCinput1(K%)
 5640 LOCALL%:ON K% GOTO5650,5660,5670,5
690,5710,5720,5730,5740,5750,5760
 5650 PROCin("Input First Sector Number"
,"(0000/2559 &000/&9FF)","09AF\&","","",
4,1,0,2559):ENDPROC
 5660 PROCin("Input Last Sector Number",
"("+F$+"/"+M$+" "+F1$+"/"+M1$+")","09AF\
&","","",4,1,F%,M%):ENDPROC
 5670 IF fls% L%=3 ELSEL%=4
 5680 PROCin("Input Sector Numbers To Be
 Saved","(0000/"+M$+" &000/"+M1$+")","09
AF\&","","",L%,1,1,M%):ENDPROC
 5690 IF fls% L%=5 ELSEL%=6
 5700 PROCin("Input Data Area To Be Save
d","(0256/"+M$+" &0A0/"+M1$+")","09AF\&"
,"","",L%,1,256,M%):ENDPROC
 5710 PROCin("Input Filename For Recover
y Area","(Max 10 chrs)","09AZ\_!","","",
10,0,0,0):ENDPROC
 5720 PROCin("Input Offset Sector","(000
0/"+A$+" &000/"+A1$+")","09AF\&","","",4
,1,0,B%):ENDPROC
 5730 PROCin("Pathname","(Max 40 chrs)
Def.: $","09AZ\*$.","","$",40,0,0,0):IF
NOT FNdirck(sr$) THEN5730 ELSEENDPROC
 5740 PROCin("Filename","(Max 10 chrs)",
"09AZ\*_!","","",10,0,0,0):ENDPROC
 5750 PROCin("Load Address","(Max 7 digi
ts) Hexadecimal Input Only/Ret -> No Cha
nge","09AF\&","",CHR$13,8,1,0,&FFFFFFF):
ENDPROC
 5760 PROCin("Exec Address","(Max 7 digi
ts) Hexadecimal Input Only/Ret -> No Cha
nge","09AF\&","",CHR$13,8,1,0,&FFFFFFF):
ENDPROC
 5770 :
 5780 DEFPROCload(F$,L%,A%)
 5800 LOCALX%,Y%:Y%=7:!&700=&712:$&712=F
$:!(&6FA+A%*4)=L%:CALL-35
 5810 ENDPROC
 5820 :
 5830 DEFPROCmsg(M%)
 5840 PROCin1:PROCcolor(1):PRINTTAB(2,29
)"    PLEASE WAIT    ";:PROCcolor(0)
 5850 IF M%=1 PRINT" Copying Selected Ar
ea Into SideWays Ram"
 5860 IF M%=2 PRINT" Saving Data Area"
 5870 ENDPROC
 5880 :
 5890 DEFPROCrecovery
 5900 LOCALA%,D%,F%,G%,H%,L%,M%,N%
 5910 PROCrec1
 5920 REPEAT
 5930 rec%=FALSE:PROCrec2
 5940 IF FNtest(69,G%)ORFNtest(69,H%) TH
EN5980
 5950 PROCrec3
 5960 IF FNtest(78,H%) PROCrec8
 5970 IF FNtest(88,H%) PROCrec9
 5980 UNTIL FNtest(69,H%)
 5990 ENDPROC
 6000 :
 6010 DEFPROCrec1
 6020 CLS:PROCstbx(0,14,79,26,0)
 6030 PROCstbx(0,16,79,0,1):PROCcolor(1)
 6040 PRINTTAB(2,15)"     Selection
":PRINTTAB(2,17)"    First Sector    "
 6050 PRINTTAB(2,19)"    Last Sector
":PRINTTAB(2,21)" Number Of Sectors "
 6060 PRINTTAB(2,23)"     Data Area
":PRINTTAB(2,25)"   Storage Type     "
 6070 PROCcolor(0)
 6080 ENDPROC
 6090 :
 6100 DEFPROCrec2
 6110 LOCALA$,I%
 6120 PROCoption(4):FORI%=15TO25STEP2:PR
INTTAB(22,I%)SPC55:NEXTI%
 6130 PROCget1("EeXxNn",6):H%=get%
 6140 IF FNtest(69,H%) ENDPROC
 6150 IF FNtest(78,H%) PROCoption(5):fls
%=TRUE ELSEPROCoption(6):fls%=FALSE
 6160 PROCget1("DdEeNnSs",1):G%=get%
 6170 IF FNtest(69,G%) ENDPROC
 6180 IF FNtest(83,G%) A$="Starting Sect
or To Last Sector":A%=1
 6190 IF FNtest(78,G%) A$="Starting Sect
or Plus Number Of Sectors":A%=2
 6200 IF FNtest(68,G%) A$="Starting Sect
or Plus Data Area":A%=3
 6210 PRINTTAB(22,15)A$
 6220 ENDPROC
 6230 :
 6240 DEFPROCrec3
 6250 LOCALsr$,A$,A1$,D$,D1$,F$,F1$,L$,L
1$,M$,M1$,N$,N1$
 6260 PROCinput1(1):F%=sr%:PROCrec4(6,5,
F%):PRINTTAB(22,17)A$"  "A1$
 6270 PROCrec4(4,3,F%):F$=A$:F1$=A1$
 6280 ON A% PROCrec5,PROCrec6,PROCrec7
 6290 N%=L%-F%+1:D%=N%*256
 6300 PROCrec4(6,5,L%):L$=A$:L1$=A1$
 6310 PROCrec4(6,5,N%):N$=A$:N1$=A1$
 6320 PROCrec4(6,5,D%):D$=A$:D1$=A1$
 6330 PRINTTAB(22,19)L$"  "L1$:PRINTTAB(
22,21)N$"  "N1$:PRINTTAB(22,23)D$"  "D1$
 6340 ENDPROC
 6350 :
 6360 DEFPROCrec4(A%,B%,C%)
 6370 A$=FNlen(A%,STR$C%,"0",0)
 6380 A1$="&"+FNlen(B%,STR$~C%,"0",0)
 6390 ENDPROC
 6400 :
 6410 DEFPROCrec5
 6420 IF fls% M%=F%+254 ELSEM%=2552+F%
 6430 IF M%>2559 M%=2559
 6440 PROCrec4(4,3,M%):M$=A$:M1$=A1$
 6450 PROCinput1(2):L%=sr%
 6460 ENDPROC
 6470 :
 6480 DEFPROCrec6
 6490 M%=2560-F%
 6500 IF fls%ANDM%>255 M%=255
 6510 IF NOTfls%ANDM%>2553 M%=2553
 6520 PROCrec4(4,3,M%):M$=A$:M1$=A1$
 6530 PROCinput1(3):sr%=sr%-1:L%=F%+sr%
 6540 ENDPROC
 6550 :
 6560 DEFPROCrec7
 6570 LOCALA%,B%,C%,D%
 6580 M%=(2560-F%)*256
 6590 IF fls%ANDM%>65280 M%=65280
 6600 IF NOTfls%ANDM%>653568 M%=653568
 6610 IF fls% A%=4:B%=3 ELSEA%=6:B%=5
 6620 PROCrec4(A%,B%,M%):M$=A$:M1$=A1$
 6630 PROCinput1(4):sr%=sr%-256
 6640 C%=sr% DIV 256:D%=sr% MOD 256
 6650 IF D%>0 L%=C%+F%+1 ELSEL%=C%+F%
 6660 ENDPROC
 6670 :
 6680 DEFPROCrec8
 6690 LOCALsr$,A$,A1$,A%,B%,C%,G%,I%
 6700 PROCmsg(1)
 6710 FORI%=4TO7:OSCLI"SRDATA "+STR$I%:N
EXTI%
 6720 *FX200,1
 6730 FORI%=F%TOL%
 6740 A%=(I%-F%)*256:PROCpv(8,I%):OSCLI(
"SRWRITE "+STR$~data%+"+100 "+STR$~A%)
 6750 NEXTI%
 6760 *FX200,0
 6770 PROCget1(CHR$13,2):OSCLI"MOUNT0"
 6780 PROCget1("FfSs",3):G%=get%
 6790 IF FNtest(70,G%) ELSE6840
 6800 PROCinput1(5):file$=sr$
 6810 PRINTTAB(22,25)"File Saving - Name
: ";file$:PROCmsg(2)
 6820 *FX200,1
 6830 OSCLI("SRSAVE "+file$+" 0+"+STR$~D
%):GOTO6930
 6840 PRINTTAB(22,25)"Sector By Sector S
aving":B%=2559-N%
 6850 PROCrec4(4,3,B%)
 6860 *FX200,0
 6870 PROCinput1(6):C%=sr%:PROCrec4(4,3,
C%)
 6880 PRINTTAB(46,25)"- Offset Sector: "
+A$+"/"+A1$:PROCmsg(2)
 6890 *FX200,1
 6900 FORI%=C%TO(C%+N%)
 6910 A%=(I%-C%)*256:OSCLI("SRREAD "+STR
$~data%+"+100 "+STR$~A%):PROCpv(10,I%)
 6920 NEXTI%
 6930 FORI%=4TO7:OSCLI"SRROM "+STR$I%:NE
XTI%
 6940 *FX200,0
 6950 IF FNtest(70,G%) PROCget1("YyNn",5
):G%=get%
 6960 IF FNtest(89,G%) PROCaddress(1)
 6970 PROCget(CHR$13,1)
 6980 IF rec% PROCrec1
 6990 ENDPROC
 7000 :
 7010 DEFPROCrec9
 7020 LOCALsr$,A$,A1$,A%,B%,C%,E%,G%,I%,
S%,T%
 7030 PRINTTAB(22,25)"Sector By Sector S
aving"
 7040 PROCcolor(1):PRINTTAB(48,17)" Loop
 ":PRINTTAB(62,17)" Of ":PRINTTAB(48,19)
"   Start Sector   ":PRINTTAB(48,21)" Se
ctor Number  ":PRINTTAB(48,23)" Loading
Pointer  ":PRINTTAB(48,25)" Saving  Poin
ter  ":PROCcolor(0)
 7050 PROCget1(CHR$13,7):OSCLI"MOUNT 0"
 7060 PROCinput1(5):file$=sr$
 7070 OSCLI("CREATE "+file$+" "+STR$~D%)
 7080 FORI%=4TO7:OSCLI"SRDATA "+STR$I%:N
EXTI%
 7090 A%=N% DIV 255:B%=N% MOD 255
 7100 IF B%>0 A%=A%+1
 7110 PRINTTAB(67,17)FNlen(2,STR$A%,"0",
0)
 7120 PROCget(CHR$13,1)
 7130 FORB%=1TOA%
 7140 PROCmsg(1):PRINTTAB(58,17)FNlen(2,
STR$B%,"0",0):PRINTTAB(67,25)"0000/&000"
 7150 C%=F%+((B%-1)*255):PROCrec4(4,3,C%
):PRINTTAB(67,19)A$"/"A1$
 7160 *FX200,1
 7170 IF B%=A% S%=(N%-(A%-1)*255)-1 ELSE
S%=254
 7180 PROCrec4(4,3,S%):PRINTTAB(67,21)A$
"/"A1$
 7190 FORI%=C%TO(C%+S%)
 7200 PROCrec4(4,3,I%):PRINTTAB(67,23)A$
"/"A1$
 7210 E%=(I%-C%)*256:PROCpv(8,I%):OSCLI(
"SRWRITE "+STR$~data%+"+100 "+STR$~E%)
 7220 NEXTI%
 7230 *FX200,0
 7240 PROCget1(CHR$13,2):PROCmsg(2)
 7250 *FX200,1
 7260 FORI%=0TOS%
 7270 PROCrec4(4,3,I%):PRINTTAB(67,25)A$
"/"A1$
 7280 E%=I%*256:OSCLI("SRREAD "+STR$~dat
a%+"+100 "+STR$~E%)
 7290 T%=7+I%+((B%-1)*255):PROCpv(10,T%)
 7300 NEXTI%
 7310 *FX200,0
 7320 IF B%<A% PROCget(CHR$13,1)
 7330 NEXTB%
 7340 FORI%=4TO7:OSCLI"SRROM "+STR$I%:NE
XTI%
 7350 PROCget1("NnYy",5):G%=get%
 7360 IF FNtest(89,G%) PROCaddress(1)
 7370 PROCget(CHR$13,1)
 7380 IF rec% PROCrec1
 7390 ENDPROC
```

512 Forum

by Robin Burton

As promised I'm going to continue with last month's topic, the redirection of standard DOS input and/or output.

MORE ON REDIRECTION

In the last Forum we examined examples of how standard input (which is normally taken from the default console input device, i.e. the keyboard) can be read instead from another source, such as a disc file, simply by modifying the line command entry. We also looked at how output to the standard console output device (usually the screen) can likewise be diverted or redirected to, for example, a disc file, a printer, or to the serial port.

As I said last month, in effect all external devices (i.e. the peripherals including the console) are regarded by the BDOS as simply input or output data channels, so let's dig a little further into this concept.

The reason I've decided to explain this part of DOS is that I always think it's easier to understand something if you have at least a reasonable idea of how it works. In this case it should help you to understand what you can and can't do with redirection, and equally important in my view, why you can do it.

The important part of the idea is the difference in the way DOS is constructed, as compared with, for example, your BBC micro's MOS. That difference is that the DOS operating system is not a single piece of code as it is in the BBC micro. The DOS system consists of two completely separate pieces of software (the BDOS and the BIOS, or the XIOS in the 512) which continually communicate with each other. I've discussed the origins of the reason for these separate elements of DOS operating systems in a recent Forum so I won't go through that again (see Vol.8 No.10).

Suffice it to say that the XIOS in the 512 (i.e. the BIOS in PCs) is the only part of the system which is concerned with physical to logical device mapping, and it actually controls the reading or writing of the data which is to be handled by external devices. The XIOS is the part of the system which is customised to suit the hardware devices supplied by each manufacturer with his system. Of course in the 512 the XIOS is the code which handles the tube interface, rather than directly dealing with hardware controllers, but in principle the concept remains the same.

DEVICE DEFINITIONS

Obviously the various devices attached to a DOS system may have differing attributes or capabilities depending on their type and purpose, so DOS must know the difference between an input only device, an output only device, and an input and output device. Even so, when data is processed by the BDOS, the source or destination of the data is irrelevant. The capabilities of devices only become a consideration when the data is processed by the XIOS (Extended Input-Output System).

The way this control is exercised within the XIOS is via a table which contains groups of bytes, with each group directly relating to each logical device (note, not physical device, as we'll see shortly) attached to, or logically capable of being attached to, the system. Although you might expect devices to be grouped as input, output (or both) DOS doesn't think like this. Strictly speaking all devices can (at least in theory) have any characteristics, so the table entry for each device must define it completely.

I should warn you that some of the following distinctions may seem rather subtle, but they are vital to the design concepts in DOS which make it so flexible. The only initial assumption in the design of DOS is that all devices attached to the system are data transfer ports of some undefined type.

The most important information, so far as DOS is concerned, is how each device processes its data, so the highest level of information, and the first thing the XIOS needs to know is to which main category of device type each logical device belongs. There are two categories, which are generally referred to as character devices, or the only alternative, block devices. You'll therefore find many DOS programming books, particularly those concerned with interrupt calls, containing references to character devices or block devices. This idea is almost self explanatory, but just to ensure that it's quite clear a couple of examples will make it more obvious.

A character device is any device that normally processes data (in either direction) on a byte by byte basis. The most obvious character device is of course the keyboard, but less obvious examples like the serial port, the screen or the printer belong in this category too. The alternative, a block device, is any device which processes data only in pre-defined 'chunks' or blocks, such as disc drives, which incidentally are the only block devices in a standard system.

Within the entry for each device, various bits are set on or off according to the detailed capabilities of the device. One bit, for example determines whether the device can transfer data into the BDOS, that is, it's an input device. Another bit is set on if the device can transfer data out of the BDOS, in which case it's naturally an output device. This is the subtle bit, and is enough to show the way DOS 'thinks' - there's no such thing as an input device, an output device or even an input/output device, there are only devices. What each device is capable of is merely a question of which of its attribute bits are set.

Other bytes within each device entry further describe relevant characteristics. For example, for the serial port, which is treated by DOS as two separate logical devices, there is the (current) data transfer speed for both (AUXIN and AUXOUT). Although there is only one physical connection on the BBC micro (two, or more on some PCs), this single physical device is mapped onto two logical devices within DOS, one input, the other output, which can be addressed quite separately.

In the case of serial devices other bytes are also needed to define the protocol, (i.e. odd or even parity, number of data bits, stop bits and so on). For other types of device these entries are irrelevant of course, so they're left empty, but the important point is that all the flags and data possible within an entry exist for all devices, even the screen and keyboard.

SYSTEM DEFAULTS

It's all very well saying that all devices are potentially capable of having all possible characteristics, but clearly in practice this isn't the case, so how are the limitations set?

In short, all this sort of thing is done for you and supplied with the system. However, it's worth saying that it may explain to some of you, now you know a bit more about it, why transferring MS-DOS from a PC to the 512 as outlined in a recent Forum (or even transferring it to another PC) is more likely to fail than to succeed.

In fact it's all part of the hardware manufacturer's job when he defines his machine's peripherals within the BIOS. He must also, in addition to supplying the data in the device table in the form that DOS expects to see it, supply the code which is needed to drive each of the different physical devices (i.e. controllers) he wishes to use in the computer. The final part of the exercise of generating a BIOS for a new machine is therefore to code the jump addresses to the relevant piece of customised code for each device type.

These code addresses, you won't be surprised to know, are also stored in a table much like the device parameters, and the inner (unchanged) part of the BIOS (or XIOS) therefore knows where each piece of code required to drive each device type is located. When the system is booted, therefore, the standard or default data for all the known devices is loaded as part of the operating system.

Items which are not (or originally, in earlier versions of DOS were not) considered to be standard devices (such as a mouse or a RAM disc) are not embedded in the operating system's defaults and do not exist unless they are added later. In many PC versions of DOS this can be accomplished by a list of entries in a file called 'CONFIG.SYS', which the operating system automatically looks for on start-up, much like the action of executing AUTOEXEC.BAT. Each entry in 'CONFIG.SYS' is in effect an additional parameter which defines the system and its facilities. By the way, for those of you new to the 512 don't look for 'CONFIG.SYS', there isn't one in DOS Plus.

THE USER INTERFACE

Anyone not familiar with my style may by now be wondering what the above sketch of a part of the 512's XIOS functions has to do with redirection, so this is where I'll tell you.

When the BDOS wishes to input or output data, it communicates with the XIOS initially by passing an identifier which defines the type of input or output required. The XIOS is responsible for identifying the device type from this and passing or reading the subsequent data to or from that device in the form (and if appropriate at the rate) the particular device dictates.

What you are doing when you redirect input or output is telling the BDOS to specify to the XIOS a different type of device than would be used by default (i.e. the standard) for the particular operation.

Discs of course are a special category of block device, since the code required to handle them is much more complicated than that needed for character devices, needing as it does to cater for several devices of the same type, each with a complex (directory) structure.

So far as character devices are concerned, there are, in the 512 and in most PCs, only two by default, CONIN and CONOUT. What happens when a facility requires input from or output to the console is that DOS exercises a bit of common sense to make life easier for the user or program.

Although either of the two console devices can frequently and conveniently be referred to by the single collective name of CON in commands, when the XIOS investigates the device table it finds that only one device of the two is capable of input, while conversely the other console device is capable only of output. The XIOS therefore reads or writes the data to or from the correct logical device without the need to be further instructed. Likewise in the case of the auxiliary port, AUX, a similar sensible choice is automatically made between the two logical devices, AUXIN and AUXOUT.

However, there is a difference between these two similar choices, and that is that in the case of the console the two logical devices are also two separate (and very different) physical devices. On the other hand, in the case of AUX the two logical devices are actually mapped onto the same single physical device.

In the case of other facilities, like PRN (apart from when you can use Ctrl-P) output to the list device requires either a program which accesses the printer explicitly (e.g. PRINT.CMD) or you must specifically name the device in a redirected command such as:

```
        TYPE FILE.TXT >PRN
```

(or if you prefer - `COPY FILE.TXT PRN`).

Next month we'll take a look at a facility which is in some ways similar to redirection, but equally is also different. It's called piping (no, we're not starting cookery lessons!).

I've also found another interesting source for shareware which I'll tell you about. This one has a particularly novel approach to the ordering procedure, and what's more the technique even saves more money than usual too.

Play: Music - The Art of Expression through Sound

Ian Waugh reviews a book for Music 5000 lovers.

Product	**Play**
Supplier	**JB Software, 20 Crawley Avenue, Wellingborough, Northants NN8 3YH. Tel. (0933) 675392**
Price	**In printed format £15.95 inc. disc On disc only £11.95**

Play is a tutorial book for the Hybrid Music System. It's not concerned with how to use the editors or with fancy programming techniques; what it sets out to do is show how you can create 'human' performances. The package includes a tutorial of almost 50 pages (this is also available as text files on disc) plus a disc of demo programs which illustrate the techniques used.

Play was written by John Bartlett who has produced about five AMPLE albums which he distributes on his own JB Software label at £3.95 each. If you've heard any of them you'll know how much 'feeling' John has managed to instil into his music. The Jazz Discs Volumes 1 and 2 are particularly impressive in this respect, although my personal favourite is Impressions which I can thoroughly recommend on both music and 'performance' levels (see review in this issue).

The approach John has adopted is to explain a range of performance techniques, and to demonstrate how he has used them in his own music. As such, *Play* is more of an examination of the working practices of John Bartlett than a specific explanation about when and where the techniques can be used. Much of the advice is of a general rather than specific nature: listen to musicians, plan your mixes carefully, understand where you want to place notes, try to understand why something doesn't work and so on. The intention is that you learn by example, although John does point out that there are many ways of accomplishing the same thing in Ample.

This approach is perfectly valid, although if you want to know just why this note should be accented or why this phrase or that should be subject to a crescendo, accelerando or particular articulation effect, you may not find all the answers here. In truth, these concepts are significantly harder to deal with (we're talking heavy music aesthetics here), and even verge on the abstract. But *Play* contains a lot of material, and you'd be hard pressed to come away from the book without a host of ideas about how you can improve your music.

It begins with some general information about conventions used in the manual followed by a couple of pages on 'Basics' and then it plunges into Timing. This looks at the timing resolution of Ample, how small timing details can't be shown on the stave, tempo and relative tempo changes including rallentandos and accelerandos, and articulation which includes staccato and legato notes. Articulation plays an incredibly important part in music phrasing, more I suspect than it's often given credit for. The Timing chapter concludes by describing how to use the WIND command to make a piece 'swing' (good, this).

The next chapter is concerned with Dynamics. It includes the use of accents and volume (and relative volume) changes, and includes crescendo and diminuendo word definitions.

The section on Mixes begins with an explanation of how to use the UMAKE command to un-make a mix in order to free voices. It also suggests that you place mix changes in words for convenience and to avoid glitches which can occur if radical mix changes are used in the PLAY word. It rounds off with a warning about how a tempo change may not have the effect you expect if it is not preceded by the correct timebase value.

Voicings, John admits, are a matter of personal taste. The program examples demonstrate a number of instruments such as a guitar, a gong and a clarinet, and there's a word (similar to the Wha instrument) which can be used for filter effects.

The Effects section explains how to use Echo to add presence, a reverb-type effect. It can also be used, of course, to produce echoes proper. There is a sustain/autopan word which uses the ACT command (this is the only example which uses 'clever programming'). This can autopan a sound or duplicate the effect of the sustain pedal on a piano. You don't have to understand how it works, which is good news, although some explanation would have been nice rather than simply being referred to the Programmer Guide. Effects concludes with a section on real-time voice modification, such as adding vibrato and pitch bend and the alteration of the waveform and pitch envelope. There are some hints about the use of string sounds and one of the demos produces an interesting guitar feedback sound.

The next two chapters (a total of five pages) tackle Accompaniments and Arrangements. These are very complex issues and although advice more specific than 'listen to the professionals' would have been nice, at least you are given the opportunity to learn from some of John's examples.

The Appendix contains a nine-page Beginners Tutorial which explains how to enter a three-part piece of music. If you haven't quite grasped the basics of entering your own pieces, this could be very helpful. Finally, there's a Comments Form which John asks you to complete and return which suggests that he is planning more projects.

Many prospective buyers, I suspect, will opt for the manual on disc. Printing it is menu-driven and very easy to follow.

In the interest of completeness, I feel obliged to point out that some of the ideas presented in the book were discussed in the three-part series, Programming in Ample in BEEBUG Vol.8 No.10 to Vol.9 No.2. But that apart, Play is the most detailed examination of music expression and interpretation on the Hybrid Music System yet to appear, and as such it joins the welcome ranks of Ample support products. If you are more a computer user than a musician, you could find the techniques described very useful indeed. I look forward to JB Software's next release. B

Inside the Mandelbrot Set (from page 12)

```
 2740 :
 2750 DEFPROCPick
 2760 VDU28,G%,31,H%,26:CLS:P%=0
 2770 REPEAT
 2780 FORI%=0TO3:COLOUR128:IFM%=1COLOUR3
ELSECOLOUR7
 2790 IFI%<>P%GOTO2810
 2800 COLOUR0:IFM%=1COLOUR131ELSECOLOUR1
35
 2810 PRINTTAB(0,I%)OP$(I%);
 2820 NEXT
 2830 J%=GET
 2840 IFJ%=139P%=P%-1:IFP%<0P%=3
 2850 IFJ%=138P%=P%+1:IFP%>3P%=0
 2860 UNTILJ%=13
 2870 IFP%=2PROCq
 2880 IFP%=3PROCBox:P%=1
 2890 ENDPROC
 2900 DEFFNr(F$,A,R,S):REM reads in Menu
 data
 2910 PRINT'"Enter new ";F$:INPUTA$:I%=L
EN(A$)
 2920 IFI%>0B=EVAL(A$):IFB<R ORB>S GOTO2
910
 2930 VDU12,26:IFI%=0THEN=A ELSE=B
 2940 :
 2950 DEFPROCq:*FX4,0
 2960 VDU22,7:PRINT"Bye"'
 2970 END
```

B

ARCHIMEDES
BBC MASTER
COMPACT
ELK
(ACP+4)

DISC 1 includes Jonathon Partington's latest release **AVON,** in which you take the part of a tourist strangely lost in The Bard's home town. Mysteriously transported into the world of His plays and players, your aim is to find out how to return to the present day. An exhaustive knowledge of the Shakespearean Canon is NOT necessary. Includes **MURDAC** free! ***£19.95***

DISC 2 features the best of Peter Killworth: much-expanded versions of **PHILOSOPHER'S QUEST & COUNTDOWN TO DOOM,** and his latest release (Part 2 of his 'Doom' trilogy) **RETURN TO DOOM.** All three games only ***£19.95***

DISC 3 includes the biggest-ever micro adventure **ACHETON,** together with a much-enhanced version of **KINGDOM of HAMIL.** Two best-sellers for ***£19.95***

DISC 4 features Peter Killworth's popular mathematics adventure game, **GIANTKILLER.** Used in schools all over Britain, this adventure for 10 to 14 year olds is a real maths challenge. Only ***£19.50***

Prices include VAT and p&p

telephone 0733 244682 24 hours

PO Box 39
Stilton
P'BORO PE7 3RL

Special Offers to BEEBUG Members - Nov 1990

BEEBUG'S OWN SOFTWARE

Code	Product	Members Price inc.VAT	Code	Product	Members Price inc.VAT
1407A	ASTAAD3 - 5" Disc (DFS)	9.95	0077B	C - Stand Alone Generator	14.25
1408A	ASTAAD3 - 3.5" Disc (ADFS)	9.95	0088C	Masterfile ADFS BBC 40 T	16.50
1404A	Beebug Applics I - 5" Disc	5.75	0089C	Masterfile ADFS BBC 80 T	16.50
1409A	Beebug Applics I - 3.5"Disc	5.75	0081C	Masterfile ADFS M128 80 T	16.50
1411A	Beebug Applics II - 5" Disc	5.75	0024C	Masterfile DFS 40 T	16.50
1412A	Beebug Applics II - 3.5" Disc	5.75	0025C	Masterfile DFS 80 T	16.50
1421B	Beebug Binder	3.99	0085C	Printwise 40 Track	22.50
1600A	Beebug magazine disc	4.75	0086C	Printwise 80 Track	22.50
1405A	Beebug Utilities - 5" Disc	5.75	0009C	Studio 8	16.50
1413A	Beebug Utilities - 3.5" Disc	5.75			
1450A	EdiKit 40/80 Track	5.75	0074C	Beebug C 40 Track	44.25
1451A	EdiKit EPROM	7.75	0075C	Beebug C 80 Track	44.25
1452A	EdiKit 3.5"	5.75	0084C	Command	29.25
0005B	Magscan Vol.1 - 8 40 Track	12.50	0073C	Command(Hayes compatible)	29.25
0006B	Magscan Vol.1 - 8 80 Track	12.50	0053C	Dumpmaster II	23.25
0011A	Magscan Update 40 track	4.75	0004C	Exmon II	24.00
0010A	Magscan Update 80 track	4.75	0087C	Master ROM	29.25

OTHER MEMBERS OFFERS

The offers listed below will only be for a limited period while stocks are available.

Acorn's 512 Co-processor

Offer price £185 (inc VAT)

We have managed to obtain the last 25 of these from Acorn. These are brand new and will probably be the last 512s ever sold. Only while stocks last.

Stock code 0231G

ViewSheet

Normal Members price £42.49 (inc VAT)
Offer price £12.50 (inc VAT)

Acorn's excellent spreadsheet for the BBC micro and Master, supplied on a 16K ROM. A very powerful and extremely useable product which will produce results ready to print or for direct merging into View text files.

Stock code 1001B

View 3 ROM

Normal Members price £51.75 (inc VAT)
Offer price £12.50 (inc VAT)

The enhanced version of the excellent View wordprocessor from Acorn. Supplied with manual,keystrip and printer driver generator on disc or tape. Many features over the original version of View.

Stock code 1022C

ViewSpell

Normal Members price £28.75 (inc VAT)
Offer price £7.00 (inc VAT)

The automatic spelling checker with a built-in 75 000 word dictionary. Supplied on ROM with full manual, examples disc and reference card. Ideal for View or ASCII text files, and can use updateable user dictionaries.

Stock code 1043B

ViewStore

Normal Members price £42.49 (inc VAT)
Offer price £13.80 (inc VAT)

ViewStore is Acorn's database manager program. It offers 40 and 80 column display, multiple indexes, detailed report generation, variable field lengths and much, much more. Excellent value for money.

Stock code 1019B

Personal Ads

BEEBUG members may advertise unwanted computer hardware and software through personal ads (including 'wants') in BEEBUG. These are completely free of charge but please keep your ad as short as possible. Although we will try to include all ads received, we reserve the right to edit or reject any if necessary. Any ads which cannot be accommodated in one issue will be held over to the next, so please advise us if you do not wish us to do this. We will accept adverts for software, but prospective purchasers should ensure that they always receive original copies including documentation to avoid any abuse of this facility.

We also accept members' Business Ads at the rate of 30p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, 117 Hatfield Road, St. Albans, Herts AL1 4JS. The normal copy date for receipt of all ads will be the 15th of each month.

BBC Model B with ROM expansion slot, twin 80T disc drives and metal monitor plinth, all in good working order, also included 20 discs full of games and childrens educational software plus 10 blank discs £265 o.n.o. Miracle Technology WS2000 modem and Command ROM £50. Tel. 081-677 0569 after 6.30pm any eve.

BBC M128 with software and mauals £250. 80186 co-processor with Gem mouse £100. Twin DS 40/80 disc drives in bridge unit £125. Philips 14" Pro SCM 852 med. res. colour monitor £150. Or the lot for £575 or exchange HF amateur RX/TX. Tel. (0872) 78020.

512 co-processor with mouse and Gem software £100. Interbase as new £40. Tel. (0525) 370686.

WANTED: Lisp for the BBC, Acornsoft publication. Tel. (0525) 370686.

BBC B 52k with Aries B20, View, Viewsheet, Viewstore ROMs fitted £250, Akhter Instruments dual DS40/80T disc drive £100, Toolkit Plus as new £10, View Printer Driver Generator £5. Tel. (0276) 65368.

BEEBUG vols 1 to 5 inclusive, all complete £1.50 per issue o.n.o., plus postage (or buyer collects - NW London). Tel. 081-455 5005.

Books; Advanced Machine Code Techniques £6, Econet Advanced User Guide £5, Advanced Disc User Guide £8. **Discs;** Repton Infinity £7, The Real You (IQ and personality tests) £7, 6 issues of Disc User (2,4,7,9,10,11) £10. **Tapes;** Speed Read course £5. **ROMs;** View Professional £30. Delta 14b joystick £5, Master dust cover £2, Dumb terminal (mono screen, keyboard, RS232) £20, Master 512, Acorn amber monitor, dual 5.25" 3.5" drives £550. Tel. 081-698 3772.

80T double sided disc drive with PSU £60, Master 128 Replay system including ROMboard 3 £40, Full set of Disc User magazines including 14 discs & binders £20, BBC Master reference manuals 1&2 £10, back issues of Acorn User 1988, 1989 including binders £15. Last Ninja & Exile discs £5 each. Dumpout 3 Utility ROM £5, all excellent condition. Tel. (0326) 240734 after 6pm.

Morley 2Mb RAM disc plus Vu-fax software. Instant loading and saving - brilliant! cost £400 - make me an offer! Morley Advanced Teletext Adaptor £40. Also 3 pages of BBC hardware, ROMs, books, games, music, utilities and programs. Most prices between 50p and £5! must sell need shelf space. Tel. 091-529 4788 anytime or send SAE for list to 26 Newark Drive, Whitburn, Sunderland, Tyne & Wear SR6 7DF.

Electron £50, Aries B32 £40, Opus Challenger 512k £100, Wordwise Plus £20, Amnesiac series 3 B offers? Tel. (0472) 840423.

Conversions; 40 to 80T, DFS to ADFS, 5.25" to 3.5". Tel. (058279) 2530.

2nd Processor 6502 with DNFS and HiBasic ROMs boxed with instruction manual £50. Tel. 081-501 2697.

WANTED: Computer Concepts Interbase with complete documentation - for a BBC Master computer. Tel. (0544) 8301.

BBC M128 as new condition with user guides etc. £300. Tel. (0254) 831576.

Two model B disc drives (3.5" and 5.25") £35 each. Many books half price. Tel. (0600) 6403.

BBC B ROMs for sale View 2.1, Viewsheet, and Dumpout 3, offers? if no offers, View & Viewsheet free if collected. (0525) 715013 (Flitwick) after 5pm or weekends.

M128 as new £280, Brother M1109 printer with Tractor feed and spare ribbons £100, Turbo co-processor £80, Z80 co-processor complete £80, Philips green screen monitor £40, AMX Superart with colourart and mouse £40, Viewplot £12, Viewstore £25, all excellent and complete (upgrading). Tel. 051-647 5367.

BBC Master in Viglen console complete with twin double sided 40/80 Opus disc drives also Viewsheet, Dumpout, Printmaster ROMs, golf, snooker etc. games £350. Kaga 625 hi-res. colour monitor £250. Philips hi-res. green screen monitor £45, all in good condition. Tel. (0268) 693770 eves.

M128, manual & guides, Interword, Interbase & manuals, Philips amber monitor, Watford 5.25" double drive, discs & box all with original packing, selling due to upgrade £550. Tel. (05242) 61321.

Acorn teletext adaptor £60, Interword £30, Spellmaster £30, View Professional £40, all boxed as new. Tel. (0684) 73173.

WANTED: Acorn Cambridge Workstation ACW443 for spares (preferably working). Tel. (0276) 33731.

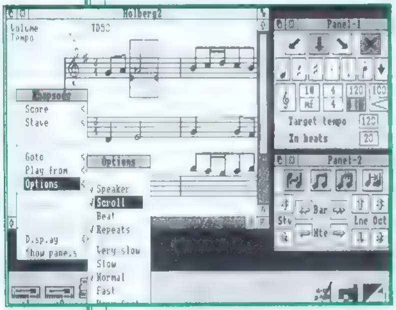
RISC USER

The Archimedes Magazine & Support Group

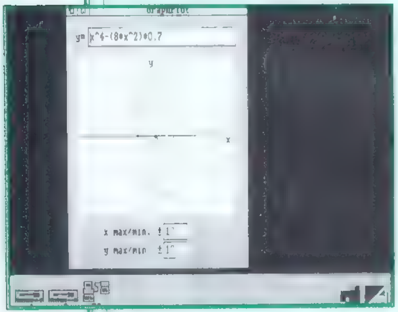
Now in its fourth year of publication, Risc User continues to enjoy the largest circulation of any magazine devoted solely to the Archimedes range of computers. It provides support for all Archimedes users at work (schools, colleges, universities, industry, government establishments) and home.

Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines.

A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer to enthusiasts and professionals at all levels.



Desktop Graph Plotter


y= x^4-(8*x^2)*0.7

Rhapsody

MVWIMPAID
A very useful application for Wimp programmers allowing pointer, window and icon parameters to be dynamically displayed and updated on screen.

ACORN'S NEW ARCHIMEDES FLAGSHIP: THE A540
An in-depth review of this fascinating new machine from Acorn.

DESKTOP GRAPH PLOTTER
A multi-tasking Wimp application which plots a graph from any equation.

A PRINTER SET-UP UTILITY
A non-Wimp program, which allows you to monitor continuously the on-line status of the printer and set other printer characteristics.

RHAPSODY
A review of the new powerful music editor from Clares.

MEGA MOUSE
A short utility which provides an intelligent response of the screen pointer in relation to mouse movements.

INTRODUCING C
Part 5 in a wide ranging new series on C, a major programming language for the Archimedes.

ASSEMBLER WORKSHOP
A major series for the more advanced ARM processor programmer. The latest one reveals some anomalies in using the ARM processor.

MASTERING THE WIMP
A major series for beginners to the Wimp programming environment. The most recent installment explains all about Implementing a Save Box.

INTO THE ARC
A regular series for beginners. The latest article looks into error handling in simple Basic programs.

As a member of BEEBUG you may extend your subscription to include RISC User for only £8.10 (overseas see table).

Phone your instructions in, or send a cheque/postal order to the address below. Please quote your **name and membership number.** When ordering by Access, Visa or Connect, please quote your card number and the expiry date.

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 8.10
Rest of Europe and Eire	£12.00
Middle East	£14.00
Americas and Africa	£15.00
Elsewhere	£17.00

RISC User, 117 Hatfield Road, St Albans, Herts AL1 4JS. Telephone (0727) 40303, FAX (0727) 860263

Our thanks to contributors for their hints in this and other recent issues of BEEBUG. Hints and tips on almost any subject relating to the BBC micro and Master series are always welcome, and we pay £5 for each one published.

SCROLLING IN EDIT

Andrew Rowland

One of the nice features of Wordwise and Interword is that the cursor remains on the same screen line, even when scrolling of text takes place, so that the eye can quickly relocate the cursor after looking away.

The same effect can be produced in the Master's Edit by positioning the cursor on the middle line and holding down Ctrl while pressing function keys f6 and f7. The text now scrolls underneath the cursor except at the very beginning of the document. Shift-f3 cancels the effect. These keys control the so-called *scroll margins* which determine how close the cursor is allowed to go to the top or bottom of the screen display before scrolling takes place automatically. By changing the margins as described they are effectively set to just the one line in the centre of the screen display.

ARABIC-ROMAN NUMBER CONVERSIONS

Ruben Hadekel

The following programs will convert Arabic numerals to Roman (program *ROMAN*) and Roman numerals to Arabic (program *ARAB*). Both are limited to numbers up to 1999 inclusive.

```
  10 REM Program ROMAN
  20 REPEAT
  30 R$=""
  40 INPUT"Enter an integer "B%
  50 IF B%>1999 PRINT"Too large":GOTO40
  60 IF B% DIV 1000=1 R$="M"
  70 C%=B% MOD 1000
  80 IF C%>899 R$=R$+"CM":GOTO 120
  90 IF C%>499 R$=R$+"D":H%=C%-500 ELSE
H%=C%
 100 IF H%>399 R$=R$+"CD":GOTO120
 110 R$=R$+STRING$(H% DIV 100,"C")
 120 D%=C% MOD 100
 130 IF D%>89 R$=R$+"XC":GOTO 170
 140 IF D%>49 R$=R$+"L":T%=D%-50 ELSE
T%=D%
 150 IF T%>39 R$=R$+"XL":GOTO170
 160 R$=R$+STRING$(T% DIV 10,"X")
 170 E%=D% MOD 10
 180 IF E%>8 R$=R$+"IV":GOTO220
 190 IF E%>4 R$=R$+"V":U%=E%-5 ELSE U%=E%
 200 IF U%>3 R$=R$+"IV":GOTO220
 210 R$=R$+STRING$(U%,"I")
 220 PRINT ''R$
 230 UNTIL FALSE
 240 END
```

```
  10 REM Program ARAB
  20 REPEAT
  30 B%=0
  40 INPUT"Enter number "N$
  50 IF RIGHT$(N$,2)="IV"
B%=4:N$=LEFT$(N$,LEN(N$)-2):GOTO90
  60 IF RIGHT$(N$,1)="I"
B%=B%+1:N$=LEFT$(N$,LEN(N$)-1):GOTO60
  70 IF RIGHT$(N$,1)="V"
B%=B%+5:N$=LEFT$(N$,LEN(N$)-1):GOTO90
  80 IF RIGHT$(N$,2)="IX"
B%=B%+9:N$=LEFT$(N$,LEN(N$)-2)
  90 IF RIGHT$(N$,1)="X"
B%=B%+10:N$=LEFT$(N$,LEN(N$)-1):GOTO90
 100 IF RIGHT$(N$,2)="XL"
B%=B%+40:N$=LEFT$(N$,LEN(N$)-2):GOTO130
 110 IF RIGHT$(N$,1)="L"
B%=B%+50:N$=LEFT$(N$,LEN(N$)-2):GOTO130
 120 IF RIGHT$(N$,2)="XC"
B%=B%+90:N$=LEFT$(N$,LEN(N$)-2)
 130 IF RIGHT$(N$,2)="C"
B%=B%+100:N$=LEFT$(N$,LEN(N$)-1):GOTO130
 140 IF RIGHT$(N$,2)="CD"
B%=B%+400:N$=LEFT$(N$,LEN(N$)-2):GOTO170
 150 IF RIGHT$(N$,2)="D"
B%=B%+500:N$=LEFT$(N$,LEN(N$)-1):GOTO170
 160 IF RIGHT$(N$,2)="CM"
B%=B%+900:N$=LEFT$(N$,LEN(N$)-2)
 170 IF RIGHT$(N$,1)="M" B%=B%+1000
 180 PRINT''B%
 190 UNTIL FALSE
 200 END
```

MORE ON ASSEMBLER MACROS

Andrew Rowland

Further to the hint on this subject in BEEBUG Vol.9 No.4, the routine given to generate errors in service ROMs will fail if the error number is zero. This can be corrected by adding the following line to the routine listed there:

```
 315 INY:LDA (zp),Y:STA &101
```

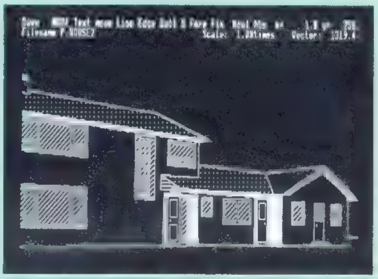
Special Offer!

on all of Best of BEEBUG software

ASTAAD

Enhanced ASTAAD CAD program for the Master, offering the following features:

- full mouse and joystick control
- built-in printer dump
- speed improvement
- STEAMS image manipulator
- Keystrips for ASTAAD and STEAMS
- Comprehensive user guide
- Sample picture files



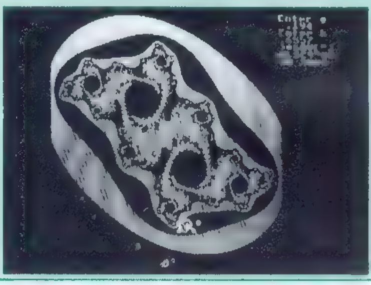
EDIKIT ROM

An indispensible utility ROM for all Basic programmers for the Master, BBC B and B+ (all with sideways RAM). It contains eleven commands to help you with the editing and development of Basic programs:

***FTEXT** (find text)	***FBASIC** (find Basic)	***FPROCFN** (find procedure/function)
***LPROC** (list procedure)	***LFN** (list function)	***LFROM** (list eight lines of a program)
***RTEXT** (replace text)	***RBASIC** (replace Basic)	
***SYSINF** (system information)	***VARLIST** (list program variables)	***FKDEFS** (function key definitions)

Incorporating the updated Basic Booster utilities:

SUPER SQUEEZE	**PARTIAL RENUMBER**	**PROGRAM LISTER**
RESEQUENCER	**SMART RENUMBER**	**TEXTLOAD AND TEXTSAVE**



Applications II Disc

Monthly Desk Diary

Crossword Editor - for designing, editing and solving crosswords

Real Time Clock - displayed on the screen

Running Four Temperatures - calibrates and plots temperatures

Label Processor - for designing and printing labels

3D Landscapes - generates three dimensional landscapes.

Foreign Language Tester - foreign characters definer

Julia Sets - extensions of the Mandelbrot set

Share Investor - assists decision making when buying and selling shares.

Applications I Disc

- Business Graphics
- Video Cataloguer
- World by Night and Day
- Phone Book
- Page Designer
- Personalised Letter-Heads
- Mapping the British Isles
- Selective Breeding
- Appointments Diary
- The Earth from Space
- Personalised Address Book

General Utilities Disc

- Printer Buffer *
- Sprite Editor/Animator
- Mode 7 Screen Editor
- Multi-Column Printing
- Epson Character Definer
- BEEBUG MiniWimp †
- ROM Filing System Generator
- ROM Controller
- Multi-Character Printer Driver for View

*** Master series only. † Requires sideways RAM.**

	Stock Code	Price		Stock Code	Price
ASTAAD (80 track DFS)	1407A	£5.95	**ASTAAD** (3.5" ADFS)	1408A	£5.95
EDIKIT (EPROM)	1451A	£7.75			
EDIKIT (40/80T DFS)	1450A	£5.75	**EDIKIT** (3.5" ADFS)	1452A	£5.75
Applications II (80 track DFS)	1411A	£4.00	**Applications II** (3.5" ADFS)	1412A	£4.00
Applications I Disc (40/80T DFS)	1404A	£4.00	**Applications I Disc** (3.5" ADFS)	1409A	£4.00
General Utilities Disc (40/80T DFS)	1405A	£4.00	**General Utilities Disc** (3.5" ADFS)	1413A	£4.00

Please add p&p - 60p for the first item and 30p for every additional item.

BEEBUG Ltd, 117 Hatfield Road, St Albans, Herts AL1 4JS. Tel. (0727) 40303 Fax. (0727) 860268

POSTBAG

A SOURCE OF MS-DOS FOR THE BEEB

In Vol.9 No.4 you published Mr Hadekel's letter calling for MS-DOS on the Master, and in your comments suggested that a full implementation might not be feasible but that programs to transfer files had been published in BEEBUG. These needed the 1770 disc controller chip, and could not format a PC disc.

Baksoft of 20 Leys Avenue, Cambridge CB4 1JF publish four suites of programs to provide transfer and print facilities for:

BBC	to and from MS-DOS
Torch CPN	to and from MS-DOS
CP/M	to and from MS-DOS
BBC	to and from CP/M

The suites are available separately at £19.50, but there is a price reduction when two or more are bought.

DFS and ADFS are catered for, and the BBC/MS-DOS program has the advantage that it will format a disc on a BBC micro for a PC, a facility which I have used regularly for some years. The MS-DOS facilities permit the use of a wide variety of disc formats to meet the requirements of differing PCs (IBM, Amstrad, Atari, Apricot, etc.).

E.L.de Bourcier

I should have remembered these products from Baksoft, as they were reviewed in BEEBUG Vol.7 No.8. The two transfer programs referred to were published in BEEBUG Vol.6 No.10 and Vol.7 No.1. Thanks to Mr de Bourcier for this information.

GETTING TRIANGULATED

There is a simple solution to Mr Holt's problem of the efficient storage of a triangular array (see Postbag Vol.9 No.5). This is to use a one dimensional array for storage. The (i,j) element from the original will then be located at j+i*(i-2)/2 in the new array.

Chris Spencer

This seems a particularly simple solution to the problem, and Mr Howard from Cumbria was another reader offering a very similar solution. As a variation, Joan Barnard from Essex also advocated a linear array to hold the data, but then the use of a second array to hold the position of the start of each row (in the first array). In this case the distance from town X to town Y is given by:

M(V(X)+(X-Y)) if X<Y, and:
M(V(Y)+(Y-X)) if X>Y.

The array M holds the original data row by row starting from element 0, and array V holds the pointers starting from V(1).

The linear array solution is simpler to implement and thus probably quicker in operation than the one I suggested last time, though indirection operators (which I used) do tend to provide the speediest access to any piece of data.

CHOOSING A DATABASE

I enjoy reading your BEEBUG surveys. Each is written by an enthusiast, but as you say (Postbag, BEEBUG Vol.9 No.5) 'each person has their own particular requirements'.

I use both Masterfile II and ViewStore, depending on the application. With Masterfile it is easy to search for and to print any entry, and easy to print selected fields. However, you have to decide at the outset how many characters to reserve for each field - too few and some entries may then need to be curtailed, too many and I may not be able to fit enough records on disc.

With ViewStore, the spreadsheet layout is particularly useful as I do not need to reserve a predetermined space. I can set how many characters to display and then scroll the information in each slot. However, the printing of selected data is not easy. A 'Select' file must be created for each set of parameters, but only one, the last one used, can be saved. Each printout takes a lot of typing, and if more than one copy is needed, the entry must be retyped in full each time.

I was asked to prepare a bibliography of books on a particular subject. I planned a database on both Masterfile and ViewStore, but decided that the more powerful ViewStore would better accommodate the 700 books and their varying title lengths. However, for my several files of addresses, Masterfile is the clear choice; it is also easy to print labels, and I can specify the number of copies of each in advance.

Michael Essex-Lopresti

BEEBUG MEMBERSHIP

Send applications for membership renewals, membership queries and orders for back issues to the address below. All membership fees, including overseas, should be in pounds sterling drawn (for cheques) on a UK bank. Members may also subscribe to RISC User at a special reduced rate.

BEEBUG SUBSCRIPTION RATES		BEEBUG & RISC USER
£16.90	1 year (10 issues) UK, BFPO, Ch.I	£25.00
£24.00	Rest of Europe & Eire	£36.00
£29.00	Middle East	£43.00
£31.00	Americas & Africa	£46.00
£34.00	Elsewhere	£51.00

BACK ISSUE PRICES (per issue)

Volume	Magazine	5"Disc	3.5"Disc
5	£1.20	£4.50	£4.50
6	£1.30	£4.75	£4.75
7	£1.30	£4.75	£4.75
8	£1.60	£4.75	£4.75
9	£1.60	£4.75	£4.75

All overseas items are sent airmail. We will accept official UK orders for subscriptions and back issues, but please note that there will be a £1 handling charge for orders under £10 which require an invoice. Note that there is no VAT in magazines.

POST AND PACKING

Please add the cost of p&p when ordering individual items. See table opposite.

Destination	First Item	Second Item
UK, BFPO + Ch.I	60p	30p
Europe + Eire	£1	50p
Elsewhere	£2	£1

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 860263
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: Alan Wrigley
Technical Assistant: Glynn Clements
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £50 per page, but please give us warning of anything substantial that you intend to write. A leaflet 'Notes to Contributors' is available on receipt of an A5 (or larger) SAE.

Please submit your contributions on disc or cassette in machine readable form, using "View", "Wordwise" or other means, but please ensure an adequate written description is also included. If you use cassette, please include a backup copy at 300 baud.

In all communication, please quote your membership number.

BEEBUG Ltd (c) 1990
Printed by Arlon Printers (0923) 268328 ISSN - 0263 - 7561

Magazine Disc

November 1990 DISC CONTENTS

INSIDE THE MANDELBROT SET - a fascinating visual display, which plots views of the interior of a Mandelbrot set.

WORDPROCESSOR STYLE INPUT - a useful routine, suitable for the BBC and Master series, which provides wordprocessing editing facility for any type of keyboard input.

BEEBUG WORKSHOP: Searching (Part 3) - this month's program demonstrates iterative searching .

PHONE CALL COSTING GOES INTERNATIONAL - an enhanced version of the very popular program from Vol.9 No.3, which includes international calls, calls to mobile phones and even freephone calls.

SIDEWAYS ROM IMAGE AUTO LOADER - a short program which automatically loads ROM images into sideways RAM.

PRACTICAL ASSEMBLER: Sorting with CALL - the complete utility, discussed over the last three months, for sorting arrays of strings, integers or reals, following CALL.

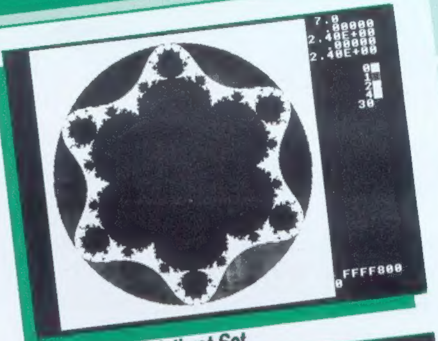
CURLY! AN EDUCATIONAL GAME - an entertaining game for children which strengthens the skills of visualisation of movement and shapes.

PRODUCING 8MM VIDEO TAPE INLAYS - The program for producing audio tape inlays, published previously, adapted to print inlays for video tapes.

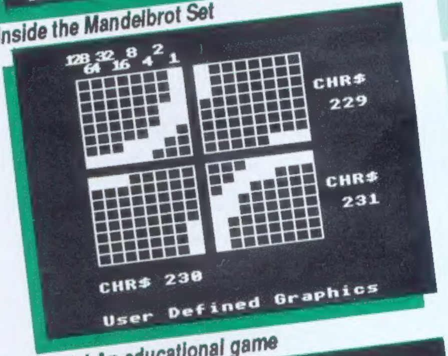
ADFS DISC SECTOR EDITOR - the complete disc recovery utility from last month with some additional menu options.

MAGSCAN DATA - bibliography for this issue (Vol.9 No.6).

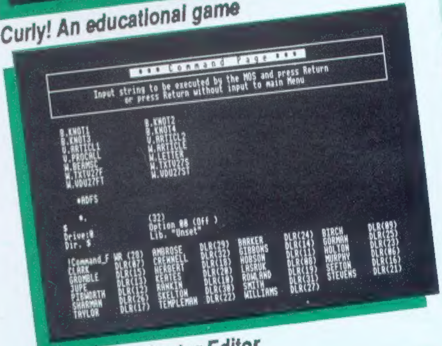
***PENTOMINOES* BONUS ITEM** - a challenging electronic version of this popular jigsaw puzzle.



Inside the Mandelbrot Set



Curly! An educational game



```
*** Command Page ***
Input string to be executed by the MOS and press Return
or press Return without input to main Menu

B.KNOT1          B.KNOT2
B.KNOT3          B.KNOT4
V.ARTICL1        V.ARTICL2
V.PROCALL        W.ARTICLE
W.BEAMSE         W.LETTER
W.TXTU27F        W.TXTU27S
W.VDU27FT        W.VDU27ST

 *ADFS

  $.             (32)
$                Option 00 (Off )
Drive:0          Lib. "Unset"
Dir. $

!Command_F WR (20)   AMBROSE    DLR(29) BARKER   DLR(24) BIRCH   DLR(09)
CLARK     DLR(07)   CREWNELL   DLR(32) GIBBONS  DLR(14) GORMAN  DLR(25)
GROMBLE   DLR(15)   HERBERT    DLR(18) HOBSON   DLR(11) HOLTON  DLR(23)
JUPE      DLR(13)   KEATES     DLR(20) LASRUK   DLR(08) MURPHY  DLR(06)
PIEWORTH  DLR(12)   RANKIN     DLR(10) ROWLAND  DLR(19) SEFTON  DLR(16)
SHARMAN   DLR(26)   SKELTON    DLR(30) SMITH    DLR(31) STEVENS DLR(21)
TAYLOR    DLR(17)   TEMPLEMAN  DLR(22) WILLIAMS DLR(27)
```

ADFS Disc Sector Editor

ALL THIS FOR £4.75 (5.25" & 3.5" DISC) + 60P P&P (30P FOR EACH ADDITIONAL ITEM)

Back issues (5.25" disc since Vol.3 No.1, 3.5" disc since Vol.5 No.1) available at the same prices.

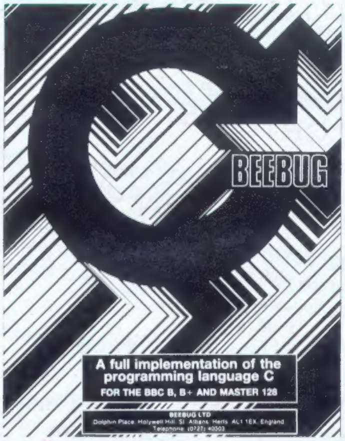
DISC (5.25" or 3.5") SUBSCRIPTION RATES	UK ONLY	OVERSEAS
6 months (5 issues)	£25.50	£30.00
12 months (10 issues)	£50.00	£56.00

Prices are inclusive of VAT and postage as applicable. Sterling only please.

BEEBUG, 117 Hatfield Road, St.Albans, Herts AL1 4JS

The C Programming Language

Beebug C for the BBC micro and Master series


BEEBUG
A full implementation of the programming language C
FOR THE BBC B, B+ AND MASTER 128
BEEBUG LTD
Dolphin Place, Holywell Hill, St. Albans, Herts. AL1 1EX. England
Telephone (0727) 40303

Beebug C is a full implementation of the exciting programming language C, which is used by many professional programmers. Although normally only available on much more powerful computers, Beebug C is a full implementation of the Kernighan and Ritchie standard for the BBC micro and Master series computers. A comprehensive set of library functions are supplied on disc, conforming to the proposed ANSI standard.

Price £51.30
(£59.00 inc.VAT)

Features include:

- Runs on a standard 32K BBC model B, B+, Master 128 and Master Compact
- 40/80 track DFS and ADFS compatible, also 3.5" Master Compact version
- Support for Acorn operating system (vdu, osbyte, mode etc.)
- Powerful command line interpreter with over 20 commands & qualifiers
- Expandable run-time library on disc containing nearly 100 functions & macros
- Full macro-handling facilities
- Debugging facilities and helpful error messages

Beebug C is supplied on two 16K ROMs and a library disc *(specify 40T or 80T)*

Stock codes:

0074C	40T disc
0075C	80T disc

"...the Beebug implementation is superior in so many respects..."
"The Beebug C system is far superior to that provided by Acorn"
"...the Beebug version has to be the best buy"
A&B COMPUTING
"...Beebug C is about five times faster at compilation and linking than the Acornsoft version..."
MICRO USER

To write C programs you will need a text editor or word processor such as View, Wordwise, InterWord etc.

Beebug C Stand Alone Generator

An add-on to Beebug C which produces stand-alone code which will run on any BBC or Master computer (memory permitting). It is supplied on a dual-format disc which also contains a number of trig functions, including:

`SIN COS TAN LOG LN EXP`

Price: £16.52 (£19.00 inc.VAT)

Stock code 0076b 5.25" 40 Track disc
Stock code 0077b 5.25" 80 Track disc